Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications

HAN CAI, JI LIN, YUJUN LIN, ZHIJIAN LIU, HAOTIAN TANG, HANRUI WANG, LIGENG ZHU, and SONG HAN, Massachusetts Institute of Technology, USA

Deep neural networks (DNNs) have achieved unprecedented success in the field of artificial intelligence (AI), including computer vision, natural language processing, and speech recognition. However, their superior performance comes at the considerable cost of computational complexity, which greatly hinders their applications in many resource-constrained devices, such as mobile phones and Internet of Things (IoT) devices. Therefore, methods and techniques that are able to lift the efficiency bottleneck while preserving the high accuracy of DNNs are in great demand to enable numerous edge AI applications. This article provides an overview of efficient deep learning methods, systems, and applications. We start from introducing popular model compression methods, including pruning, factorization, quantization, as well as compact model design. To reduce the large design cost of these manual solutions, we discuss the AutoML framework for each of them, such as neural architecture search (NAS) and automated pruning and quantization. We then cover efficient on-device training to enable user customization based on the local data on mobile devices. Apart from general acceleration techniques, we also showcase several task-specific accelerations for point cloud, video, and natural language processing by exploiting their spatial sparsity and temporal/token redundancy. Finally, to support all these algorithmic advancements, we introduce the efficient deep learning system design from both software and hardware perspectives.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Natural language processing**; **Computer vision**; • **Hardware** → *Reconfigurable logic and FPGAs*; • **Computer systems organization** → **Neural networks**; *Parallel architectures;*

Additional Key Words and Phrases: Efficient deep learning, TinyML, model compression, AutoML, neural architecture search

ACM Reference format:
Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Haotian Tang, Hanrui Wang, Ligeng Zhu, and Song Han. 2022. Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications. *ACM Trans. Des. Autom. Electron. Syst.* 27, 3, Article 20 (February 2022), 50 pages.
https://doi.org/10.1145/3486618

1 INTRODUCTION

Deep neural networks (DNNs) have revolutionized the field of **artificial intelligence (AI)** and have delivered impressive performance in computer vision [118, 155, 263], natural language processing [22, 71, 271, 290], and speech recognition [69, 122, 327]. They can be applied in various

All student authors have contributed equally to this work and are listed in the alphabetical order.
Authors' addresses: H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han (corresponding author), Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA, 02139; emails: {hancai, jilin, yujunlin, zhijian, kentang, hanrui, ligeng, songhan}@mit.edu.

1084-4309/2022/02-ART20
https://doi.org/10.1145/3486618

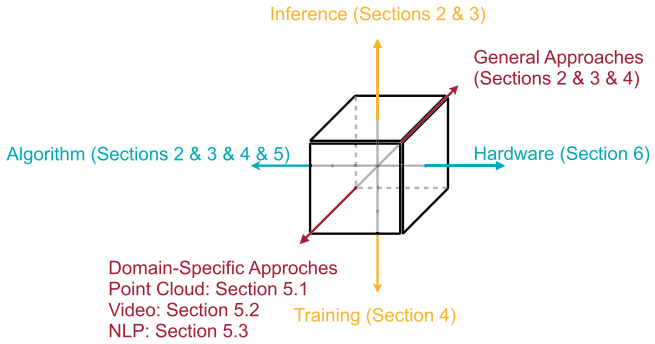


Fig. 1. Left: Spectrum of efficient deep learning solutions (from inference to training, from algorithm to software/hardware system, from general to domain-specific). Right: Overview of article organization.

real-world scenarios, such as mobile phones [131, 217, 328], self-driving cars [5, 20, 59, 194], and smart hospitals [116, 183, 337]. However, their superior performance comes at the cost of high computational complexity. For instance, the state-of-the-art machine translation model [290] requires more than 10 G **multiply-and-accumulates (MACs)** to process a sentence of only 30 words; the popular LiDAR perception model [56] needs more than 2000 G MACs per second (i.e., 10 frames).

Such a high computational cost is far beyond the capabilities of most mobile devices, ranging from the vehicles to the mobile phones and the **Internet of Things (IoT)** devices, since their hardware resources are tightly constrained by the form factor, battery, and heat dissipation. These computation workloads, however, cannot be delegated to the cloud server, as they can be very sensitive to the latency (i.e., autonomous driving) and/or privacy (i.e., healthcare) [199, 359]. Therefore, efficient deep learning is in great demand to lift the roadblock for mobile AI applications.

To accelerate the neural network inference, researchers have proposed a variety of model compression techniques, including pruning [115, 121, 195], low-rank factorization [149, 331, 351], and quantization [62, 114, 133]. Besides building upon existing large models, researchers have also explored designing efficient neural networks directly from scratch, including MobileNets [125, 250], ShuffleNets [202, 350], and SqueezeNets [202, 350]. These solutions usually require considerable human efforts, since there are a bunch of knobs that need to be tuned jointly to achieve the optimal performance: i.e., the pruning ratio and the quantization bitwidth of each layer. To this end, there have been many explorations to use **automated machine learning (AutoML)** to provide push-the-button solutions to free the human from the time-consuming design process, such as **neural architecture search (NAS)** [29, 106, 187, 277, 365], automated pruning [120, 196, 335], and automated quantization [298, 299, 305]. However, the benefit of AutoML does not come for free, as it will increase the carbon footprints significantly: i.e., Evolved Transformer [264] produces the lifetime carbon dioxide emissions of five U.S. cars (in Figure 2). To achieve green and sustainable AI, researchers have proposed to efficiently search efficient neural architectures [25, 295], which can achieve the same level of accuracy while reducing the carbon footprints by orders of magnitudes.

Besides the inference, neural network training can be very expensive as well, which hinders the on-device training and, consequently, the user customization on the mobile devices. To tackle this, researchers have proposed various memory-efficient training algorithms, such as gradient checkpointing [44], activation pruning [65], and low-bit quantization [355]. In most use cases, the mobile model only needs to be fine-tuned a little bit on the local user data to provide the specialization. Hence, another stream of research attempts to improve the efficiency of transfer learning [26, 214]. Lately, researchers have also introduced the federated learning [151] to aggregate the users' trained models without compromising the privacy.

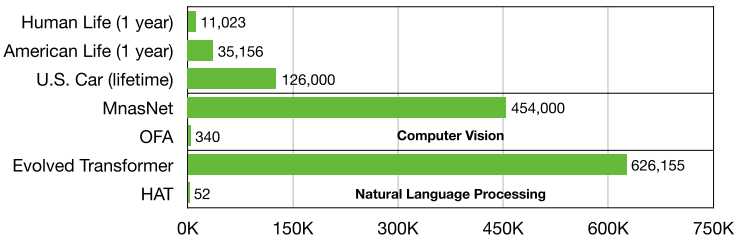


Fig. 2. Deep learning can introduce large carbon footprints: i.e., Evolved Transformer [264] produces the lifetime carbon emissions of five cars. Thus, efficient deep learning is critical for green and sustainable AI.

Apart from general accelerations that can be applied to any task in principle, there have been extensive investigations in domain-specific accelerations. In this article, we will focus on point cloud processing, video understanding, and natural language processing, because they are widely used in mobile applications, such as autonomous driving and mobile vision/NLP. On the one hand, they are much more computationally expensive than conventional 2D vision due to their large memory footprint. On the other hand, they also provide unique opportunities for acceleration by exploiting and removing the spatial and temporal redundancies: spatial redundancy (point clouds), temporal redundancy (videos), and token-level redundancy (natural languages).

Not all algorithmic improvements, however, can be translated into the measured acceleration on hardware. For instance, sparse and low-bit computations (which are introduced by fine-grained pruning and quantization) are not natively supported by the general-purpose inference library (i.e., cuDNN [52]) as well as hardware (i.e., CPUs, GPUs). This gap has been gradually bridged by recent efforts on designing specialized software systems [43, 72, 137, 138] and hardware systems [4, 48, 113, 219, 256, 297, 352]. The specialized software systems explore intra and inter operation parallelism inside the neural network and optimize the computation graph and memory scheduling via heuristic rules and even learning-based methods. The specialized hardware systems directly support the sparsity of the pruned networks and mixed-precision quantization from the hardware architecture level. The specialization of software and hardware systems opens up a new design space orthogonal to the algorithm space, which can be further exploited to unlock the unfulfilled potentials of specialization. Researchers thus have explored diverse co-design solutions, such as automatically assigning the computation resource on various platforms for NN models [140, 144, 211, 212], automatically sizing the hardware architecture [334, 365], and even jointly searching the neural network and accelerator design including connectivity between processing elements and loop scheduling [181].

There are many existing surveys related to model compression [50, 54, 68], automated machine learning [78, 119, 316], efficient hardware architecture design [272], and optimizations for specific tasks [282]. This article aims to cover a wider spectrum of methods and applications for efficient deep learning: from manual to automated, from new primitives/operations design to design space exploration, from training to inference, from algorithm to hardware, and from general-purpose to application-specific optimizations. We believe that this survey article will provide a more holistic view of this field.

The remainder of the article will be structured as follows (Figure 1):

- Section 2 discusses various model compression methods, including pruning, low-rank factorization, quantization, knowledge distillation, and compact model design.
- Section 3 studies AutoML frameworks for model compression and neural architecture search.
- Section 4 describes efficient on-device training (general/transfer learning techniques).

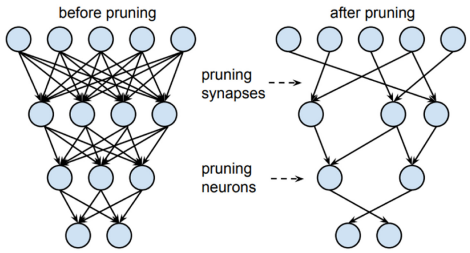


Fig. 3. Parameter pruning [115].

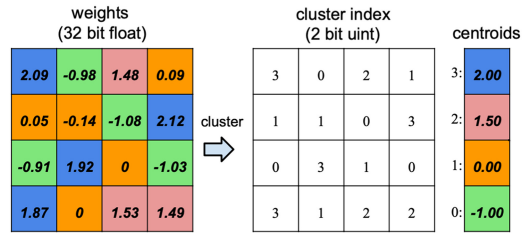


Fig. 4. k-means weight quantization [109].

- Section 5 studies application-specific acceleration for point clouds, videos, and languages.
- Section 6 introduces the efficient software/hardware design for deep learning.

2 MODEL COMPRESSION

2.1 Parameter Pruning

Deep neural networks are usually over-parameterized. Pruning removes the redundant elements in neural networks to reduce the model size and computation cost (Figure 3).

Granularity. Pruning can be performed at different *granularities*:

- *Fine-grained pruning* removes individual elements from the weight tensor. Early approaches include Optimal Brain Damage [163] and Optimal Brain Surgeon [117], which reduce the number of connections based on the Hessian of the loss function. Han et al. [115] propose a three-step method, train-prune-retrain, to prune the redundant connections in a deep neural network. It reduces the number of parameters of AlexNet by a factor of 9×, and VGG-16 by 13×, with no loss of accuracy. Srinivas et al. [265] propose a data-free pruning method to remove the redundant neurons. In fine-grained pruning, the set of weights to be pruned can be chosen arbitrarily, and it can achieve a very high compression ratio on CNN [115], RNN [92], LSTM [112], and Transformers [51] without hurting accuracy.
- *Pattern-based pruning* is a special kind of fine-grained pruning that has better hardware acceleration with compiler optimization [203, 216, 279]. It assigns a fixed set of masks to each 3×3 kernel. The number of the masks is usually limited (4–6) to ensure hardware efficiency. Despite the intra-kernel fine-grained pruning pattern, pattern-based pruning can be accelerated with compiler optimization by reordering the computation loops, reducing the control-flow overhead.
- *Coarse-grained pruning* or *structured pruning* removes a regular tensor block for better hardware efficiency. Depending on the block size, entire vectors, kernels [216], or channels [121, 171, 213, 314] are removed. Coarse-grained pruning like channel pruning can bring direct hardware acceleration on GPUs using standard deep learning libraries, but it usually comes at noticeable accuracy drop compared with fine-grained sparsity [171]. Pruning using a smaller granularity usually brings smaller accuracy drop at the same compression rate.

Hardware Acceleration. Regular pruning schemes are more hardware-friendly, making it easier for inference acceleration on existing hardware like GPUs, while more irregular pruning schemes better preserve the accuracy at the same compression rate. With specialized hardware accelerators [48, 49, 112, 113, 340, 347] and compiler-based optimization techniques [203, 216], it is also possible to gain a considerable acceleration speed for more irregular pruning methods.

Importance Criteria. After choosing a pruning granularity, determining which weights to be pruned is also essential to the pruned models' performance. There have been several importance criteria heuristics to estimate the importance of each weight *after* the model is trained; the less important weights are pruned according to the criteria. The most straightforward heuristic is based on the magnitude: i.e., absolute weight values $|w|$ [109, 115], where the weights of larger magnitude are considered as more important. It also extends to coarse-grained pruning like channel pruning, where the norm of tensor $\|\mathbf{W}\|_2$ is used as the criterion. Other criteria include second-order derivatives (the Hessian of the loss function) [117, 163], loss-approximating Taylor expansion [213], and output sensitivity [79]. Recently, Frankle and Carbin propose Lottery Ticket Hypothesis [86] to find a sparse sub-network within the dense, randomly initialized deep networks *before* training, which can be trained to achieve the same accuracy. Experiments show that the method can find sparse sub-networks with less than 10%–20% of weights while reaching the same level of accuracy on MNIST [162] and CIFAR [154]. It is later scaled up to larger-scale setting (i.e., ResNet-50 and Inception-v3 on ImageNet), where the sparse sub-network can be found at the early phase of training [87] instead of initialization.

Training Methods. Directly removing the weights in a deep neural network will significantly hurt the accuracy at a large compression ratio. Therefore, some training/fine-tuning is needed to recover the performance loss. Fine-tuning can be done after pruning to recover the performance drop [121]. It can be extended to iterative pruning [114, 115], where multiple iterations of pruning and fine-tuning are performed to further boost the accuracy. To avoid incorrect pruning of weights, dynamic pruning [105] incorporates connection splicing into the whole process and makes it as a continual network maintenance. Runtime pruning [178] chooses the pruning ratio according to each input sample, assigning a more aggressive pruning strategy for easier samples to achieve a better accuracy-computation tradeoff. Another implementation trains compact DNNs using sparsity constraints. The sparsity constraints are usually implemented using L_0, L_1, or L_2-norm regularization applied to the weights, which are added to the training loss for joint optimization. Han et al. [115] apply L_1/L_2 regularization to each individual weight during training. Lebedev et al. [161] apply group sparsity constraints on convolutional filters to achieve structured sparsity.

2.2 Low-rank Factorization

Low-rank factorization uses matrix/tensor decomposition to reduce the complexity of convolutional or fully connected layers in deep neural networks. The idea of using low-rank filters to accelerate convolution has been long investigated in signal processing area.

The most widely used decomposition is Truncated **Singular Value Decomposition (SVD)** [94], which is effective for accelerating fully connected layers [70, 93, 331]. Given a fully connected layer with weight $W \in \mathbb{R}^{m \times k}$, the SVD is defined as $W = USV^{\mathrm{T}}$, where $U \in \mathbb{R}^{m \times m}, S \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times k}$. S is a diagonal matrix with the singular values on the diagonal. If the weight falls in a low-rank structure, then it can be approximated by keeping only t largest entries of S, where $t \ll \min(m, k)$. The computation Wx can be reduced from $O(mk)$ to $O(mt + tk)$ for each sample.

For 4D convolutional weights, Jaderberg et al. [134] propose to factorize $k \times k$ kernels into $1 \times k$ and $k \times 1$ kernels, which is also adopted in Inception-V3 design [274]. Zhang et al. [351] propose to factorize a convolution weight of $n \times c \times k \times k$ into $n' \times c \times k \times k$ and $n \times n' \times 1 \times 1$, where $n' \ll n$. **Canonical Polyadic (CP)** decomposition can be used to decompose higher dimensional kernels like convolutional weights [160]. It computes a low-rank CP-decomposition of the 4D convolution kernel tensor into a sum of a small number of rank-one tensors. At inference time, the original convolution is replaced with a sequence of four convolutional layers with smaller kernels.

Kim et al. [149] use Tucker Decomposition (the higher-order extension of SVD) to factorize the convolutional kernels, getting higher compression ratio compared to using SVD.

2.3 Quantization

Network quantization compresses the network by reducing the bits per weight required to represent the deep network (Figure 4). The quantized network can have a faster inference speed with hardware support.

Rounding Schemes. To quantize a full-precision weight (32-bit floating-point value) to lower precision, rounding is used to map the floating-point value into one of the quantization buckets.

- Early explorations [95, 109, 319] apply *k-means clustering* to find the shared weights for each layer of a trained network: i.e., all the weights that fall into the same cluster will share the same weight. Specifically, when partitioning n original weights $W = \{w_1, w_2, \ldots, w_n\}$ into k clusters $C = \{c_1, c_2, \ldots, c_k\}, n \gg k$, we minimize the **within-cluster sum of squares (WCSS)**:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2. \tag{1}$$

  This can be combined with pruning and Huffman coding to perform model compression [109], which can compress the model size of VGG-16 by 49× with no loss of accuracy.
- *Linear/uniform* quantization [133] directly rounds the floating-point value into the nearest quantized values after range truncation, and the gradient is propagated using STE approximation [18]. Suppose the clipping range is $[a, b]$, and the number of quantization levels is n, the forward of quantizing floating-point value x into quantized value q is:

$$q = \text{round}\left(\frac{\text{clamp}(x, a, b) - a}{\text{s}(a, b, n)}\right) \text{s}(a, b, n) + a, \tag{2}$$

  where $\text{clamp}(x, a, b) = \min(\max(x, a), b)$ and $\text{s}(a, b, n) = (b - a)/(n - 1)$. The back-propagation gradient is approximated by $\partial \mathcal{L}/\partial q = \partial \mathcal{L}/\partial x$. For relatively high-bit quantization (i.e., 8), a and b can be set as the minimum and maximum value of the weight tensor. It is also beneficial to choose the optimal a, b values that reach the minimum Kullback-Leibler divergence between the floating-point weights and quantization weights. Apart from using the truncation values, some work [355] uses activation function such as tanh to map the range of the weights into $[-1, 1]$, making it easier for quantization.

Bit-precision. We can trade off the model size and accuracy by using different bit-precisions. A lower bit-precision can lead to a smaller model size, but it may come at the cost of accuracy drop. Full-precision networks use FP32 for both weights and activations. Half-precision networks use FP16 to reduce the model size by half. INT8 quantization for both weights and activations [133] is widely used for integer-arithmetic-only inference, which can be accelerated on CPUs and GPUs.

Lower precision models include Ternary Weight Networks [167], where the weights are quantized to $\{-1, 0, +1\}$ or $\{-E, 0, +E\}$ (where E is the mean absolute weight value). Trained Ternary Quantization [357] uses two learnable full-precision scaling coefficients W_l^p and W_l^n for each layer l and quantizes the weights to $\{-W_l^n, 0, +W_l^P\}$. The extreme case for low-bit quantization is binary weight neural networks (i.e., BinaryConnect [61], BinaryNet [62], XNOR [241]), where weights are represented with only 1 bit. The binary weights/activations are usually learned directly during

network training. BinaryConnect [61] discusses both deterministic binarization:

$$w_b = \text{sign}(w) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{3}$$

and stochastic binarization:

$$w_b = \begin{cases} +1 & \text{with probability } p = \sigma(w), \\ -1 & \text{with probability } 1 - p, \end{cases} \tag{4}$$

where σ is the "hard sigmoid" function:

$$\sigma(x) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right). \tag{5}$$

Quantization Schemes. For quantization of higher precisions (i.e., INT8), it is possible to perform *post-training quantization*, where the weights and activations are quantized after the full-precision model training. The quantization range for activations is determined by computing the distribution on training set. Applying post-training INT8 quantization usually leads to minor or no loss of accuracy. Recent work [14] also studies the post-training quantization of INT4 models.

Quantization-aware training can reduce the quantization accuracy loss by emulating inference-time quantization during training [133]. The forward pass during training is consistent with testing time, which helps the on-device deployment. During training, the "fake quantization operator" is injected into the convolutional layers, and the batch normalization [132] layers are folded.

Both post-training quantization and quantization-aware training require the access to the training data to get a good quantization performance, which is not always feasible on some privacy-sensitive applications. *Data-free quantization* aims to reduce the bit-precisions with no access to the training data. Nagel et al. [215] propose to perform INT8 quantization in a data-free manner equalizing the weight ranges in the network. ZeroQ [30] optimizes for a Distilled Dataset to match the statistics of batch normalization across different layers of the network for data-free quantization.

2.4 Knowledge Distillation

Knowledge distillation (KD) [23, 123] can transfer the "dark knowledge" learned in a large model (denoted as the teacher) to a smaller model (denoted as the student) to improve the performance of the smaller one. The small model is either a compressed model or a shallower/narrower model. Bucilua et al. [23] achieve the goal by training the student network to match output logits; Hinton et al. [123] introduce the idea of temperature in the softmax output and trained the student to mimic the softened distribution of the teacher model's softmax output. KD shows promising results in various image classification tasks despite the simple implementation.

Apart from the final output, intermediate activations also contain useful information. FitNet [245] trains the student to mimic the full feature map of the teacher model through regression. **Attention Transfer (AT)** [344] transfers the attention map of the activation from teacher to student, which is the summation of the feature map across channel dimension. Both methods require the intermediate activation to share the same spatial resolution, which limits the choice of the student model.

KD-based method is also applicable to other applications beyond classification, including object detection [39], semantic segmentation [192], language modeling [251], and image synthesis [172].

Table 1. Summarized Results of Manually Designed CNN Architectures on the ImageNet Dataset

	#Params (M)	#MACs (M)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
AlexNet [155]	60	720	57.2	80.3
GoogleNet [273]	6.8	1,550	69.8	89.5
VGG-16 [263]	138	15,300	71.5	–
ResNet-50 [118]	25.5	4,100	76.1	92.9
SqueezeNet [130]	1.2	1,700	57.4	80.5
MobileNetV1 [125]	4.2	569	70.6	89.5
MobileNetV2 [250]	3.4	300	72.0	–
MobileNetV2-1.4 [250]	6.9	585	74.7	–
ShuffleNetV1-1.5× [350]	3.4	292	71.5	–
ShuffleNetV2-1.5× [202]	3.5	299	72.6	–
ShuffleNetV2-2× [202]	7.4	591	74.9	–

2.5 Manual Neural Architecture Design

Besides compressing an existing deep neural network, another widely adopted approach to improving efficiency is to design new neural network architectures. A CNN model typically consists of convolution layers, pooling layers, and fully connected layers, where most of the computation comes from convolution layers. For example, in ResNet-50 [118], more than 99% **multiply-accumulate operations (MACs)** are from convolution layers. Therefore, designing efficient convolution layers is the core of building efficient CNN architectures. There are three widely used efficient convolution layers, including 1×1/pointwise convolution, group convolution, and depthwise convolution:

- **1×1 Convolution.** 1×1 convolution (also called pointwise convolution) is a special kind of standard convolution layer, where the kernel size K is 1. Replacing a $K \times K$ standard convolution layer with a 1×1 convolution layer will reduce #MACs and #Params by K^2 times. In practice, as the 1×1 convolution itself cannot aggregate spatial information, it is combined with other convolution layers to form CNN architectures. For example, 1×1 convolution is usually used to reduce/increase the channel dimension of the feature map in CNN.
- **Group Convolution.** Different from 1×1 convolution that reduces the cost by decreasing the kernel size dimension, group convolution reduces the cost by decreasing the channel dimension. Specifically, the input feature map is split into G groups along the channel dimension. Each group is then fed to a standard $K \times K$ convolution of size $(O_c/G) \times (I_c/G) \times K \times K$. Finally, the outputs are concatenated along the channel dimension. Compared to a standard $K \times K$ convolution, #MACs and #Params are reduced by G times in a group convolution.
- **Depthwise Convolution.** The number of groups G is an adjustable hyperparameter in group convolutions. A larger G leads to lower computational cost and fewer parameters. An extreme case is that G equals the number of input channels I_c. In that case, the group convolution layer is called a depthwise convolution. While the computational cost of a depthwise convolution is lower than a group/normal convolution, its modeling capacity is lower than the group/normal convolution. In practice, depthwise convolution is usually used for edge devices (i.e., mobile), while group/normal convolution is usually used for cloud devices (i.e., GPU).

Based on these efficient convolution layers, there are three representative manually design efficient CNN architectures, including SqueezeNet [130], MobileNets [125, 250], and ShuffleNets [202, 350].

- **SqueezeNet.** SqueezeNet [130] targets extremely compact model sizes for mobile applications. It has only 1.2 million parameters but achieves an accuracy similar to AlexNet (Table 1). SqueezeNet has 26 convolution layers and no fully connected layer. The last feature map goes through a global average pooling and forms a 1,000-dimension vector to feed the softmax layer. SqueezeNet has eight *Fire* modules. Each fire module contains a squeeze layer with 1×1 convolution and a pair of 1×1 and 3×3 convolutions. SqueezeNet achieves a top-1 accuracy of 57.4% and a top-5 accuracy of 80.5% on ImageNet [67].
- **MobileNets.** MobileNetV1 [125] is based on a building block called *depthwise separable convolution*, which consists of a 3×3 depthwise convolution layer and a 1×1 convolution layer. The input image first goes through a 3×3 standard convolution layer with stride 2, then 13 depthwise separable convolution blocks. Finally, the feature map goes through a global average pooling and forms a 1,280-dimension vector fed to the final fully connected layer with 1,000 output units. With 569M MACs and 4.2M parameters, MobileNetV1 achieves 70.6% top-1 accuracy on ImageNet (Table 1). MobileNetV2 [250], an improved version of MobileNetV1, also uses 3×3 depthwise convolution and 1×1 convolution to compose its building blocks. Unlike MobileNetV1, the building block in MobileNetV2 has three layers, including a 3×3 depthwise convolution layer and two 1×1 convolution layers. The intuition is that the capacity of depthwise convolution is much lower than the standard convolution, thus more channels are needed to improve its capacity. From the cost perspective, the #MACs and #Params of a depthwise convolution only grow linearly (rather than quadratically like standard convolution) as the number of channels increases. Thus, even having a large channel number, the cost of a depthwise convolution layer is still moderate. Therefore, in MobileNetV2, the input feature map first goes through a 1×1 convolution to increase the channel dimension by a factor called *expand ratio*. Then the expanded feature map is fed to a 3×3 depthwise convolution, followed by another 1×1 convolution to reduce the channel dimension back to the original value. This structure is called *inverted bottleneck* and the block is called *mobile inverted bottleneck block*. Besides the mobile inverted bottleneck block, MobileNetV2 has another two improvements over MobileNetV1. First, MobileNetV2 has skip connections for blocks in which the stride is 1. Second, the activation function of the last 1×1 convolution in each block is removed. Combining these improvements, MobileNetV2 achieves 72.0% top-1 accuracy on ImageNet with only 300M MACs and 3.4M parameters (Table 1).
- **ShuffleNets.** ShuffleNetV1 also utilizes 3×3 depthwise convolution rather than standard convolution in its building blocks, similar to MobileNets. Besides, ShuffleNetV1 introduces two new operations, pointwise group convolution and channel shuffle. The pointwise group convolution's motivation is to reduce the computational cost of 1×1 convolution layers. However, it has a side effect: A group cannot see information from other groups. This will significantly hurt accuracy. The channel shuffle operation is thus introduced to address this side effect by exchanging feature maps between different groups. After shuffling, each group will contain information from all groups. On ImageNet, ShuffleNetV1 achieves 71.5% top-1 accuracy with 292M MACs (Table 1). Following a similar idea, in ShuffleNetV2, the input feature map is divided into two groups at the beginning of each building block to reduce the computational cost. One group goes through the convolution branch that consists of a 3×3 depthwise convolution layer and two 1×1 convolution layers. The other group goes through a skip connection when the stride is 1 and goes through a 3×3 depthwise separable convolution when the stride is 2. The outputs are concatenated along the channel dimension, followed by a channel shuffle operation to exchange information between groups. With 299M MACs, ShuffleNetV2 achieves 72.6% top-1 accuracy on ImageNet (Table 1).

2.6 Future Directions

Most of the existing work studies model compression using hardware-unrelated metrics such as MACs or model size, or using direct metrics like latency given a pre-defined hardware/software system. There could be a large potential on co-designing the model compression scheme and compiler optimization or hardware design. However, model compression usually starts from a pre-defined/pre-trained deep network and compresses it for a more efficient deployment. The compression space is largely based on the pre-defined network architecture. Therefore, if the network is not originally designed for a specific edge hardware (i.e., deploy ResNets on mobile phones), then compression may not be able to close the huge gap compared to other efficient network designs like MobileNets. In such cases, designing a new network architecture from scratch could bring larger efficiency improvement. We will discuss more about automatic network architecture design with **Neural Architecture Search (NAS)** in the next section.

3 AUTOMATED COMPRESSION AND NEURAL ARCHITECTURE SEARCH

The success of the aforementioned model compression strategies and efficient neural network architectures relies on hand-crafted heuristics that require domain experts to explore the large design space, trading off among model size, latency, energy, and accuracy. This is time-consuming and sub-optimal. In this section, we describe automated methods for tackling this challenge.

3.1 Automated Model Compression

Model compression methods can improve the efficiency of the deployed models. However, the performance of model compression is largely affected by the hyperparameters. For example, different layers in deep networks have different capacities and sensitivities (i.e., the first layer in CNN is usually very sensitive to pruning). Therefore, we should apply different pruning ratios for different layers of the network to achieve the optimal performance. The design space is so large that human heuristic is usually sub-optimal, and manual model compression is time-consuming. To this end, automated model compression is proposed to find good compression policy without human effort.

Automated Pruning. Conventional model pruning techniques rely on hand-crafted features and require domain experts to explore the large design space, trading off among model size, speed, and accuracy, which is usually sub-optimal and time-consuming. **AutoML for Model Compression (AMC)** [120] leverages reinforcement learning to efficiently sample the design space and find the optimal pruning policy for a given network. The reward is calculated as a function of accuracy and FLOP. AMC outperforms manually tuned compression baselines like Han et al. [111] in a fully automated manner. A recent work MetaPruning [196] first trains a PruningNet, a kind of meta network, which is able to generate weight parameters for any pruned structure given the target network, and then uses it to search for the best pruning policy under different constraints. The meta network can be used to directly measure the compression accuracy without fine-tuning.

Automated Quantization. Mixed-precision quantization also requires extensive effort deciding the optimal bit-width for each layer to achieve the best accuracy-performance tradeoff. **Hardware-Aware Automated Quantization (HAQ)** [298] is proposed to automate the process. HAQ leverages the reinforcement learning to automatically determine the quantization policy. Compared with conventional methods, HAQ is fully automated and can specialize the quantization policy for different neural network architectures and hardware architectures. Done et al. [74] propose a second-order quantization method **Hessian AWare Quantization (HAWQ)** for mixed precision quantization. HAWQ allows for the automatic selection of the relative quantization precision of each layer, based on the layer's Hessian spectrum. It also shows superior performance when applied to large language models [258].

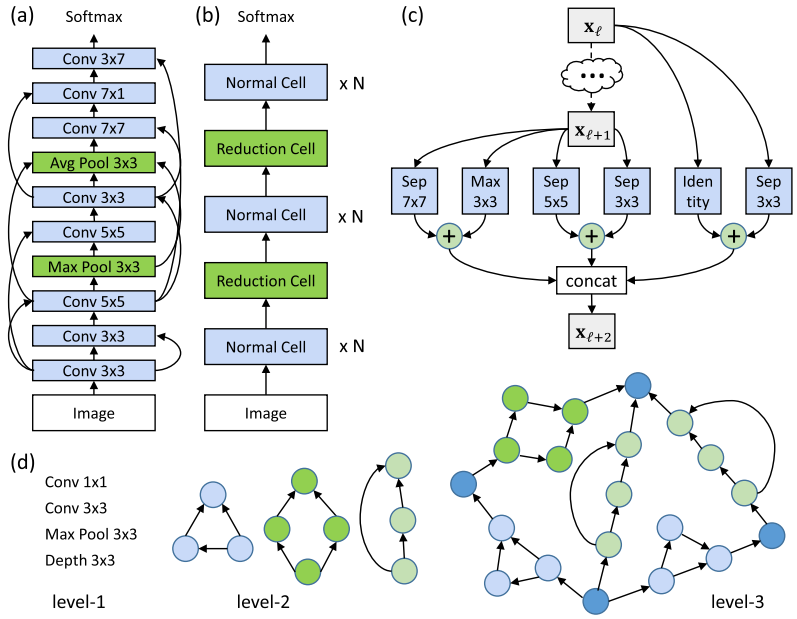


Fig. 5. Design spaces for neural architecture search [68]: (a) network-level search space [365]; (b) cell-level search space [366]; (c) an example of learned cell structure [185]; (d) three-level hierarchical search space [186].

3.2 Automated Neural Architecture Design

Neural Architecture Search (NAS) refers to automatic methods for neural network architecture design. In the conventional NAS formulation [365], designing neural network architectures is modeled as a sequence generation problem, where an auto-regressive RNN controller is introduced to generate neural network architectures. This RNN controller is trained by repeatedly sampling neural network architectures, evaluating the sampled neural network architectures and updating the controller based on the feedback.

To find a good neural network architecture in the vast search space, this process typically requires to train and evaluate tens of thousands of neural networks (i.e., 12,800 in Zoph et al. [366]) on the target task, leading to prohibitive computational cost (10^4 GPU hours). To address this challenge, many techniques are proposed that try to improve different components of NAS, including search space, search algorithm, and performance evaluation strategy.

Search Space. All NAS methods need a pre-defined search space that contains basic network elements and how they connect with each other. For example, the typical basic elements of CNN models consist of (1) convolutions [243, 366]: standard convolution (1×1, 3×3, 5×5), asymmetric convolution (1×3 and 3×1, 1×7 and 7×1), depthwise-separable convolution (3×3, 5×5), dilated convolution (3×3); (2) poolings: average pooling (3×3), max pooling (3×3); (3) activation functions [240]. Then these basic elements are stacked sequentially [12] with identity connections [365]. The full network-level search space grows exponentially as the network deepens. When the depth is 20, this search space contains more than 10^{36} different neural network architectures [366].

Instead of directly searching on such an exponentially large space, restricting the search space is a very effective approach for search acceleration. Specifically, researchers propose to search for

Table 2. Results of Different Search Algorithms for Neural Architecture Search

	Algorithm	GPU Days	CIFAR-10		ImageNet	
			#Params (M)	Accuracy (%)	#MACs (M)	Accuracy (%)
MetaQNN [12]	Q-learning	100 (–)	–	93.1	–	–
NAS [365]	REINFORCE	22,400 (K40)	37.4	96.4	–	–
EAS [24]	REINFORCE	10 (GTX 1080)	10.7	96.6	–	–
BlockQNN [354]	Q-learning	96 (Titan X)	39.8	96.5	–	75.7
NASNet [366]	PPO	2,000 (P100)	3.3	96.6	564	74.0
PNASNet [185]	SMBO	250 (P100)	3.2	96.6	588	74.2
AmoebaNet [243]	EA	3,150 (K40)	34.9	97.9	570	75.7
DARTS [187]	Gradient	4 (GTX 1080Ti)	3.3	97.2	574	73.3

basic building cells (Figure 5(b)) that can be stacked to construct neural networks, rather than the entire neural network architecture [354, 366]. As such, the architecture complexity is independent of the network depth, and the learned cells are transferable across different datasets. It enables NAS to search on a small proxy dataset (i.e., CIFAR-10) and then transfer to another large-scale dataset (i.e., ImageNet) by adapting the number of cells. This proxy-based approach greatly reduces the search cost. However, it usually provides worse performances than directly searching on the target large-scale dataset. Within the cell, the complexity is further reduced by supporting hierarchical topologies [186] or increasing the number of elements (blocks) in a progressive manner [185].

Search Algorithm. NAS methods usually have two stages at each search step: (1) the generator produces an architecture and then (2) the evaluator trains the network and obtains the performance. As getting the performance of a sampled neural network architecture involves training a neural network, which is very expensive, search algorithms that affect the sample efficiency play an important role in improving the search speed of NAS. Most of the search algorithms used in NAS fall into five categories: random search, **reinforcement learning (RL)**, evolutionary algorithms, Bayesian optimization, and gradient-based methods. Among them, RL, evolutionary algorithms, and gradient-based methods empirically provide the most competitive results (Table 2).

RL-based methods model the architecture generation process as a Markov Decision Process, treat the accuracy of the sampled architecture as the reward, and update the architecture generation model with RL algorithms, including Q-learning [12, 354], REINFORCE [365], and PPO [366]. Instead of training an architecture generation model, evolutionary methods [186, 243] maintain a population of neural network architectures. This population is updated through mutation and recombination. While both RL-based methods and evolutionary methods optimize neural network architectures in the discrete space, DARTS [187] proposes continuous relaxation of the architecture representation. The output y is modeled as the weighted sum of candidate operations' outputs ($\{o_i(x)\}$):

$$y = \sum_i \alpha_i o_i(x), \qquad \alpha_i \geq 0, \sum_i \alpha_i = 1, \tag{6}$$

where α_i is the architecture parameter representing the probability of choosing candidate operation o_i. Such continuous relaxation allows optimizing neural network architectures in the continuous space using gradient descent, which greatly improves the search efficiency. Besides the above techniques, the search efficiency can be improved by exploring the architecture space with network transformation operations, starting from an existing network, and reusing the weights [24, 28, 77].

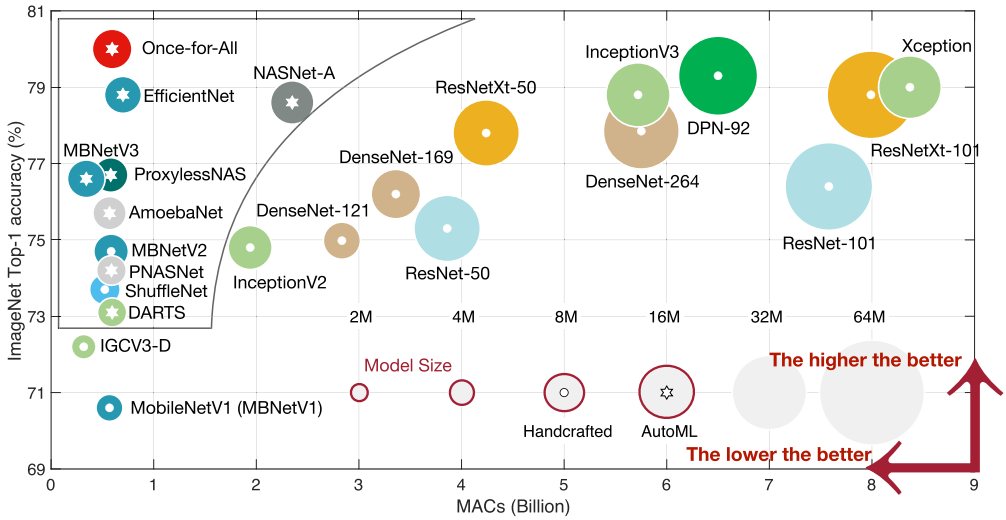


Fig. 6. Summarized results of auto-designed and human-designed CNN models on the ImageNet dataset [25].

Performance Evaluation. To guide the search process, NAS methods need to get the performances (typically accuracy on the validation set) of sampled neural architectures. The trivial approach to get these performances is to train sampled neural network architectures on the training data and measure their accuracy on the validation set. However, it will result in excessive computational cost [243, 365, 366]. This motivates many techniques that aim at speeding up the performance evaluation step. Alternatively, the evaluation step can also be accelerated using Hypernetwork [21], which can directly generate weights of a neural architecture without training it. As such, only a single Hypernetwork needs to be trained, which greatly saves the search cost. Similarly, one-shot NAS methods [29, 187, 226] focus on training a single super-net, from which small sub-networks directly inherit weights without training cost.

Auto-designed vs. Human-designed. Figure 6 reports the summarized results of auto-designed and human-designed CNN models on ImageNet. NAS saves engineer labor costs and provides better CNN models over human-designed CNNs. Besides ImageNet classification, auto-designed CNN models have also outperformed manually designed CNN models on object detection [46, 91, 278, 366] and semantic segmentation [40, 184].

Hardware-aware Neural Architecture Search. While NAS has shown promising results, achieving significant MACs reduction without sacrificing accuracy, in real-world applications, we care about the real hardware efficiency (i.e., latency, energy) rather than #MACs. Unfortunately, reduction in #MACs does not directly translate to measured speedup. Figure 7 shows the comparison between auto-designed CNN models (NASNet-A and AmoebaNet-A) and human-designed CNN model (MobileNetV2-1.4). Although NASNet-A and AmoebaNet-A have fewer MACs, they actually run slower on hardware than MobileNetV2-1.4.

It is because #MACs only reflects the computation complexity of convolutions. Other factors such as data movement cost, parallelism, and cost of element-wise operations that significantly affect real hardware efficiency are not taken into consideration. This problem motivates hardware-aware NAS techniques [29, 277, 317] that directly incorporate hardware feedback into the architecture search process. For example, in MNAS [277], each sampled neural network

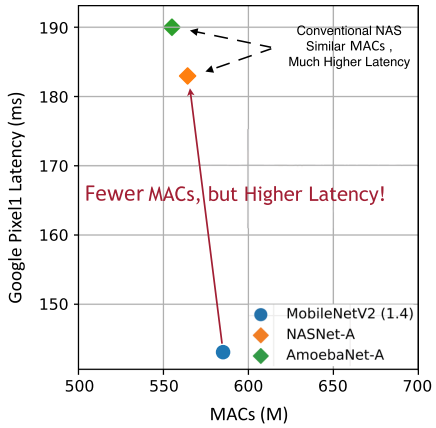


Fig. 7. #MACs does not reflect the real hardware efficiency. NASNet-A and AmoebaNet-A (auto-designed CNN models) have fewer MACs than MobileNetV2-1.4 (human-designed CNN model). However, they run slower than MobileNetV2-1.4 on the Google Pixel 1 phone.

architecture is measured on the target hardware to collect its latency information, in addition to its accuracy information. A multi-objective reward is defined based on accuracy and latency: reward = accuracy $\times$ (latency/T)$^{\omega}$, where T is the target latency and ω is a hyperparameter.

Latency Prediction. Measuring the latency on-device is accurate but not ideal for scalable neural architecture search. There are two reasons: (i) *Slow.* As suggested in TensorFlow-Lite, we need to average hundreds of runs to produce a precise measurement, approximately 20 seconds. This is far more slower than a single forward/backward execution. (ii) *Expensive.* A lot of mobile devices and software engineering work are required to build an automatic pipeline to gather the latency from a mobile farm. Instead of direct measurement, an economical solution is to build a prediction model to estimate the latency [29]. In practice, this is implemented by sampling neural network architectures from the candidate space and profiling their latency on the target hardware platform. The collected data is then used to build the latency prediction model. For hardware platforms that sequentially execute operations, like mobile device and FPGA, a simple latency lookup table that maps each operation to its estimated latency is sufficient to provide very accurate latency predictions [25, 29]. For hardware platforms where this sequential execution assumption does not hold, such as GPU and CPU, we can train a neural network to predict the latency [295].

Diverse Deployment Scenarios. Although specialized CNNs are superior over non-specialized ones, designing specialized CNNs for every scenario is still difficult, either with human-based methods or hardware-aware NAS, since such methods need to *repeat* the network design process and *retrain* the designed network from scratch for *each* case. Their total cost grows linearly as the number of deployment scenarios increases (Table 3), which will result in excessive energy consumption and CO_2 emission [268]. It makes them unable to handle the vast amount of hardware devices (23.14 billion IoT devices till 2018) and highly dynamic deployment environments (different battery conditions, different latency requirements). To handle this challenge, one promising direction is to train a single neural network that supports diverse architecture configurations, amortizing the training cost. Wu et al. [322], Liu et al. [188], and Wang et al. [307] propose to learn a controller or gating modules to adaptively drop layers; Huang et al. [128] introduce early-exit branches in the computation graph; Lin et al. [178] adaptively prune channels based on the input feature map; Kuen et al. [156] introduce stochastic downsampling point to reduce the feature map size adaptively; Slimmable Nets [339, 341] propose to train a model to support multiple width multipliers

Table 3. Summarized Results of Different Neural Architecture Search Frameworks [25]

	Accuracy	#MACs	Mobile Latency	Searching Cost (GPU Hours)	Training Cost (GPU Hours)	Total Cost ($N = 40$)		
						GPU Hours	CO_2e (lbs)	AWS Cost
MobileNetV2 [250]	72.0%	300M	66ms	0	$150N$	6K	1.7K	$18.4K
MobileNetV2 #1200 [250]	73.5%	300M	66ms	0	$1200N$	48K	13.6K	$146.9K
NASNet-A [366]	74.0%	564M	–	$48{,}000N$	–	1,920K	544.5K	$5875.2K
DARTS [187]	73.1%	595M	–	$96N$	$250N$	14K	4.0K	$42.8K
MnasNet [277]	74.0%	317M	70ms	$40{,}000N$	–	1,600K	453.8K	$4896.0K
FBNet-C [317]	74.9%	375M	–	$216N$	$360N$	23K	6.5K	$70.4K
ProxylessNAS [29]	74.6%	320M	71ms	$200N$	$300N$	20K	5.7K	$61.2K
SinglePathNAS [106]	74.7%	328M	–	$288 + 24N$	$384N$	17K	4.8K	$52.0K
AutoSlim [338]	74.2%	305M	63ms	180	$300N$	12K	3.4K	$36.7K
MobileNetV3-Large [124]	75.2%	219M	58ms	–	$180N$	7.2K	1.8K	$22.2K
OFA [25]	76.0%	230M	58ms	40	1,200	1.2K	0.34K	$3.7K
OFA #75 [25]	76.9%	230M	58ms	40	$1{,}200 + 75N$	4.2K	1.2K	$13.0K

In this table, the accuracy is evaluated on the ImageNet dataset, the mobile latency is measured with the Pixel 1 phone, the CO_2 emission ("CO_2e") is calculated following Strubell et al. [268], and the AWS cost is estimated based on the price of on-demand P3.16xlarge instances.

(i.e., four different global width multipliers), building upon existing human-designed neural networks (i.e., MobileNetV2 0.35, 0.5, 0.75, 1.0). Recently, Cai et al. [25] introduce techniques to train a more powerful **once-for-all (OFA)** network, which enables a much more diverse architecture space (depth, width, kernel size, and resolution) and a significantly larger number of architectural settings.

Design the Search Space. Search space design is crucial to the final NAS performance. Existing network design space is usually derived from manual network design (i.e., the search space used in ProxylessNAS [29] is derived from MobileNetV2 [250]). Recent works investigate the design of search space itself. Radosavovic et al. [238] propose to design a network design space that parametrizes populations of networks, which consist of simple, regular networks called RegNet. The core insight of such parametrization is to represent the widths and depths by a quantized linear function. The optimized network design space contains good network architectures that can be found by random search. Recently, Lin et al. [175] propose TinyNAS, a two-stage neural architecture search method for memory-constrained deployment on **microcontrollers (MCUs)**. Due to the lack of search space design for MCUs, TinyNAS first optimizes the search space itself to improve the performance of neural architecture search.

AutoML for TinyML. TinyML is a new frontier for edge deep learning computing. AutoML-based methods have been applied to TinyML area. SpArSe [81] employs a Bayesian optimization framework that jointly selects model architecture and optimizations such as pruning to meet memory constraints. MCUNet [175] co-designs the efficient neural architecture and efficient compiler/runtime to enable ImageNet-scale applications on off-the-shelf microcontrollers. MicroNet [13] observes that, on average, model latency varies linearly with model operation (op) count for models in the search space. It then employs differentiable NAS to search for models with low memory usage and low op count. Rusci et al. [247] use **reinforcement learning (RL)** to find a good mixed-precision quantization policy to help fit an ImageNet model on MCUs.

3.3 Joint Compression and Neural Architecture Search

As designing efficient neural network architectures and model compression are orthogonal to each other, in practice, we can combine these two techniques to further boost efficiency.

Sequential Optimization. A straightforward approach to doing this is applying these techniques separately (Figure 8, upper). For example, Cai et al. [27] employ a sequential AutoML pipeline to

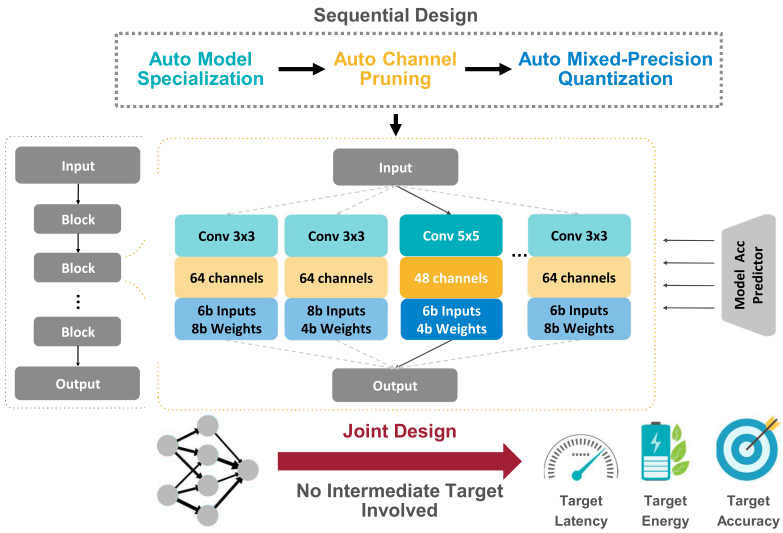


Fig. 8. Sequential optimization framework (upper) and joint optimization framework (lower) for designing efficient neural architecture and searching best pruning/quantization policy [305].

accelerate neural network inference on hardware, which starts with searching an efficient neural network architecture with hardware-aware NAS [29], then applies automated channel pruning [120] and mixed-precision quantization [298] to compress the searched neural network. A critical drawback of this straightforward approach is that optimizing in separate stages will lead to sub-optimal results: i.e., the best network architecture for the full-precision model is not necessarily optimal after pruning and quantization. Besides, each step has its own optimization objective (i.e., accuracy, latency, energy) and thus requires considerable human efforts and computational cost to tune the intermediate targets.

Joint Optimization. Unlike sequential optimization approaches, a joint optimization approach tackles this problem in an end-to-end manner, which is preferred (Figure 8, lower). However, directly extending existing AutoML techniques to the joint model optimization setting can be problematic. First, the joint search space is much larger (multiplicative) than the stage-wise search, making the search difficult. Pruning and quantization usually require a time-consuming fine-tuning process to restore accuracy [298, 335], which dramatically increases the search cost.

APQ [305] proposes to use the *quantization-aware accuracy predictor* to accelerate this joint optimization. The predictor takes the model architecture and the quantization scheme as input and can quickly predict its accuracy. Instead of fine-tuning the pruned and quantized network to get the accuracy, it uses the estimated accuracy generated by the predictor, which can be obtained with negligible cost (since the predictor requires only a few FC layers). Training this quantization-aware accuracy predictor requires collecting a lot of (quantized model, quantized accuracy) data points, where getting each of the data points could be quite expensive: (1) We need to train the network to get the initial FP32 weights (2) and further fine-tune to get the quantized INT8 weights to evaluate the accuracy. Both stages are quite expensive, requiring hundreds of GPU hours.

To reduce the cost of stage 1, APQ employs a pre-trained once-for-all network [25] that supports all sub-networks while achieving on-par or even higher accuracy compared to training from scratch. To reduce the cost of stage 2, APQ proposes a *predictor-transfer technique*. APQ first trains an FP32 model accuracy predictor using the cheap (FP32 model, FP32 accuracy) data points

collected with the weight-sharing once-for-all network (evaluation only, no training required). Then APQ transfers the predictor to the quantized model domain by fine-tuning it on a small number of expensive (quantized model, quantized accuracy) data points. The transfer technique dramatically improves the sample efficiency on the quantized network domain and reduces the overall cost to train the predictor. Under the same latency/energy constraint, APQ can attain better accuracy than the sequentially optimized model (74.1% vs. 71.8%). This is reasonable, since the per-stage optimization might fall into local optimal results while the joint design approach does not.

3.4 Limitations and Future Directions

Methodologically, current AutoML research mainly focuses over searching on restricted design spaces, such as searching several pre-defined architectural hyperparameters (i.e., depth, width, kernel size, resolution) based on existing human-designed neural network architectures. While this can simplify the task and reduce the search cost, it limits the optimization headroom, making it impossible to discover novel primitive operations or building blocks. Besides, different hardware platforms typically require different search spaces. For example, depthwise convolution provides a very good tradeoff between latency and accuracy on mobile platforms, while it is less effective for GPUs, since it cannot fully utilize the hardware parallelism. Thus, an important future research direction is to break this limitation and extend AutoML to more general and diverse design spaces. This also requires designing better AutoML algorithms to handle more complicated design spaces. Besides, another future research direction is to extend AutoML to more machine learning applications, which requires combining AutoML techniques with domain-specific insights.

4 EFFICIENT ON-DEVICE LEARNING

In real-world edge AI applications, intelligent edge devices keep collecting *new* data through the sensor every day while being expected to provide high-quality and customized services without sacrificing privacy. These pose new challenges to efficient AI techniques that could not only run inference but also continually adapt the models to newly collected data (i.e., on-device learning).

Though on-device learning can enable many appealing applications, it is an extremely challenging problem. First, edge devices are *memory-constrained*. For example, a Raspberry Pi 1 Model A only has 256 MB of memory, which is sufficient for inference, but by far insufficient for training (Figure 9), even with a lightweight neural network (MobileNetV2 [250]). Furthermore, the memory is shared by various on-device applications (i.e., other deep learning models) and the operating system. A single application may only be allocated a small fraction of the total memory, which makes this challenge more critical. Second, edge devices are *energy-constrained*. DRAM access consumes two orders of magnitude more energy than on-chip SRAM access. The large memory footprint of activations cannot fit into the limited on-chip SRAM, thus it has to access DRAM. For instance, the training memory of MobileNetV2, under batch size 16, is close to 1 GB, which is by far larger than the SRAM size of an AMD EPYC CPU (Figure 9), not to mention lower-end edge platforms. If the training memory can fit on-chip SRAM, then it will drastically improve the speed and energy efficiency.

In this section, we describe efficient training techniques towards the goal of efficient on-device learning. The key difference between inference and training is that training requires storing all intermediate activations for back-propagation while inference does not. Besides, the activation size grows linearly w.r.t. the training batch size. As a result, activation arises to be the major bottleneck for training. For example, under batch size 16, the activation size of ResNet50 [118] is 13.9× larger than its parameter size (Figure 10). Therefore, reducing the activation size is the core objective of most efficient training techniques.

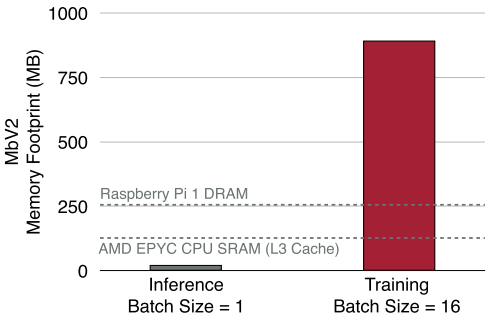


Fig. 9. The memory footprint required during training is much larger than inference.

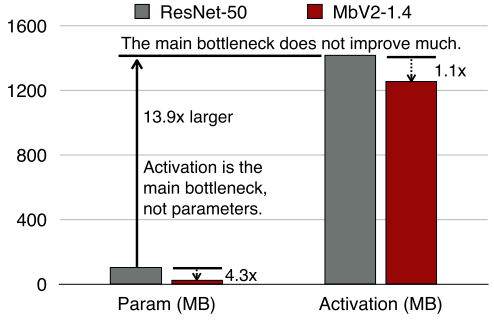


Fig. 10. Memory cost comparison between ResNet-50 and MobileNetV2-1.4 (under batch size 16). Recent advances in efficient model design only reduce the size of parameters, but the activation size, which is the bottleneck for training, does not improve much.

4.1 General Efficient Training Techniques

Gradient Checkpointing. One typical approach to reducing the training activation size is to discard a subset of intermediate activations [44, 101]. During the backward pass, the discarded intermediate activations will be re-computed to obtain the gradient. We can divide the neural network into k segments, and the activation size of training this model is reduced from $O(n)$ to $O(n/k) + O(k)$. Setting $k = \sqrt{n}$, the activation size becomes $O(2\sqrt{n})$.

Activation Pruning. Besides dropping intermediate activations, another approach to reduce the activation size is activation pruning. Liu et al. [189] build a dynamic sparse computation graph to prune activations during training. Similar to weight pruning, non-critical neurons are removed to reduce the memory footprint and save computational cost. These non-critical neurons can be selected according to their output activations. If the output activation has a small or negative value, then it will be small/zero after ReLU. Thus, removing them does not significantly affect the final result. This process is input-dependent and does not remove any neurons permanently.

Low-bit Training. Apart from inference, *training* with quantized weights, activations, and gradients can reduce the cost of deep learning training. Training with mixed 16-bit and 32-bit floating-point types in a model has been widely supported by deep learning frameworks such as TensorFlow and PyTorch. Hardware like NVIDIA Volta GPU architecture also paves the way for mixed-precision training. Additionally, custom data formats such as BFloat16 and TensorFloat-32 allow for a larger dynamic range to preserve accuracy. With techniques like loss scaling, such mixed-precision training can reduce the memory consumption and improve training speed with no loss of accuracy. DoReFa-Net uses 1-bit weights, 2-bit activations, and 6-bit gradients for faster training and inference, which can obtain comparable accuracy compared to FP32 for AlexNet [155] on ImageNet [67]. Lin et al. [182] stochastically binarize the weights to reduce the time of FP multiplication in training.

Gradient Compression. For large-scale distributed training, the quantization is no longer enough, as the maximum saving is 32× (from FP32 to a single bit), while the gap between high-end networking and normal one is 100× (100 Gbps infini-band vs. 1 Gbps Ethernet). To improve the scalability of multi-node training, a more effective method is needed to reduce bandwidth requirements. Thus, methods on reducing the transferred bits have been proposed, such as gradient quantization [253, 315] and gradient compression [180, 270, 312]. By applying quantization or

compression before exchanging gradients, the transferred bits can be reduced by a large margin (up to 600× as in DGC [180]). The accuracy can be well preserved with warm-up training and error compensation.

4.2 Efficient Transfer Learning

The aforementioned efficient training techniques mainly target the general case where neural networks are trained from scratch. This learning paradigm is suitable for cloud-based learning, where the number of data samples is sufficient. However, for on-device learning scenarios where the number of data samples is limited, it will be difficult to train deep neural networks from scratch. Alternatively, we can transfer a pre-trained neural network to the target on-device task (i.e., transfer learning), which is much more data-efficient. This also allows us to take advantage of existing powerful pre-trained neural networks [22, 71], which take extensive human efforts to design and huge computational resources to train.

Specifically, neural networks pre-trained on large-scale datasets (i.e., ImageNet [67]) are widely used as a fixed feature extractor for transfer learning, then only the last layer needs to be fine-tuned [37, 73, 89, 254]. This approach does not require to store the intermediate activations of the feature extractor, and thus is memory-efficient. However, the capacity of this approach is limited, resulting in poor accuracy, especially on datasets [204] whose distribution is far from ImageNet (i.e., only 45.9% Aircraft top1 accuracy achieved by Inception-V3 [214]). Alternatively, fine-tuning the full network can achieve better accuracy [63, 152]. But it requires a vast memory footprint and hence is not friendly for training on edge devices. Recently, Mudrakarta et al. [214] and Frankle et al. [88] propose to only update parameters of the **batch normalization (BN)** [132] layers, which greatly reduces the number of trainable parameters. Unfortunately, parameter-efficiency does not translate to memory-efficiency (Figure 10, right). It still requires a large amount of memory (i.e., 326 MB under batch size 8) to store the input activations of the BN layers.

Instead of focusing on reducing the number of trainable parameters, Cai et al. [26] propose ***tiny transfer learning*** **(TinyTL)**, which targets reducing the training memory footprint. The key insight of TinyTL is that the intermediate activations are only required to update weights, while updating biases does not need them. TinyTL proposes to freeze the weights of the pre-trained feature extractor while only updating the biases. To compensate for the capacity loss due to freezing the weights, TinyTL introduces lite residual learning that exploits a new class of generalized memory-efficient bias modules to refine the intermediate feature maps. On cars, TinyTL provides the same level of accuracy as fine-tuning the full network while reducing the memory cost by 4.6× [26].

4.3 Federated Learning

The privacy of personal data is gaining growing attention recent years and it leads to increasing demand of training without breaking privacy. Federated learning [208] is such a protocol that allows multi clients to jointly train a model without explicitly sharing their data. While it is common to have many edge devices in deployments, federated learning provides a way to utilize all of them and address the concerns of security, as the local data never leaves the client. There have been applications such as keyboard content suggestions [116] and medical treatments analysis [142].

Different from clusters equipped with high-end network infrastructures, the edge devices are usually connected with a less powerful network (i.e., Wi-Fi). In this case, the bandwidth is low and the latency is high and conventional methods scale poorly. To eliminate the bottleneck, federated average [208], gradient compression [31, 180], and quantization [133] greatly reduce the transferred bits to reduce the bandwidth requirements and delayed updated [359, 361] deals with the latency issue.

4.4 Discussions and Future Directions

TinyTL [26] reduces the transfer learning memory from more than 250 MB to only 16 MB, making it promising for in-memory computing for training. For single-device cases, reducing the training memory footprint and the energy consumption is the key challenge for efficient on-device learning. For better efficiency, a promising direction is to put the whole training process into the cache (SRAM), which is much more energy-efficient and faster than DRAM training. To approach this goal, we need to combine advances from both the algorithm domain and the hardware/software system domain. For multi-device cases, the key challenge is the connection quality. Previous distributed training is designed for high-end networking infrastructure like Infini-Band, but the networking of edge devices (i.e., Wi-Fi) can be unstable and slow. It is also important to protect the data ownership, as direct message exchange may leak private user data. To achieve the target, we need to consider from both the system and privacy perspectives.

5 DOMAIN-SPECIFIC OPTIMIZATION

Apart from task-agnostic accelerations that can be applied to any domain, there have also been extensive investigations in optimizing for specific tasks. In this article, we will focus on point cloud, video, and natural language processing. On the one hand, they are more computationally expensive than conventional 2D vision due to their large memory bandwidth. On the other hand, they also offer unique opportunities for acceleration: spatial sparsity (point clouds), temporal redundancy (videos), and token redundancy (natural languages).

5.1 Efficient Point Cloud Processing

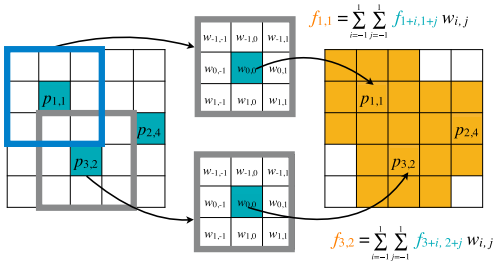
Recently, emerging applications such as AR/VR and autonomous driving have been developing rapidly. In these applications, it is crucial to efficiently process 3D data, usually point clouds out of LiDAR sensors. However, such goal is particularly challenging in 3D, since 3D deep learning models are usually an order of magnitude more expensive than image CNNs given similar input size. Different from image data that is represented as dense matrices or tensors, 3D point clouds are usually represented as a sparse set of points: $\boldsymbol{x} = \{\boldsymbol{x}_k\} = \{(\boldsymbol{p}_k, \boldsymbol{f}_k)\}$, where $\boldsymbol{p}_k$ is the 3D coordinate of the kth point, and $\boldsymbol{f}_k$ is the feature corresponding to $\boldsymbol{p}_k$. Due to the sparse nature of 3D point clouds, they can not be effectively processed by conventional image CNNs, but by specialized DNNs composed of point cloud convolution operations. The major challenges for point cloud convolution are two-fold: large memory footprint introduced by the additional spatial dimension, and irregular memory access pattern introduced by sparse data format.

Point Cloud Convolution. The general form of point cloud convolution can be written as:

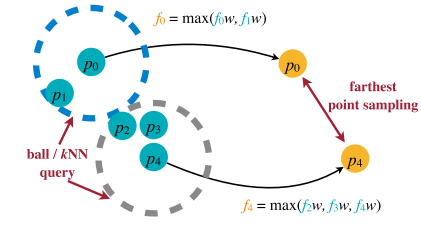
$$\boldsymbol{y}_k = \sum_{\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x}_k)} \mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i) \times \mathcal{F}(\boldsymbol{x}_i), \tag{7}$$

During the convolution, we iterate the center $\boldsymbol{x}_k$ over the entire input. For each center, we first index its neighbor $\boldsymbol{x}_i$ in neighborhood $\mathcal{N}(\boldsymbol{x}_k)$, then convolve the neighboring features $\mathcal{F}(\boldsymbol{x}_i)$ with the kernel $\mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i)$, and finally produce the corresponding output $\boldsymbol{y}_k$.

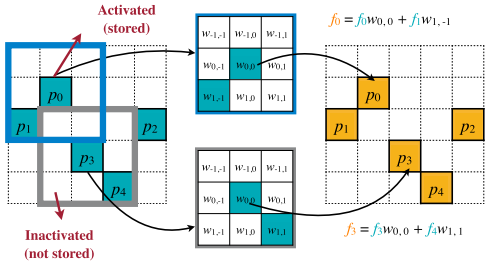
- **Voxel-based Convolution.** Early research on 3D deep learning relies on volumetric representation to process point cloud data [58, 207, 232, 323, 356] (Figure 11(a)). The point cloud coordinates $\boldsymbol{p}_k$ are first quantized into integers, and the point cloud is converted to the dense tensor representation via voxelization. Maturana et al. [207] propose to generalize 2D CNNs to vanilla 3D CNNs to further extract features from the voxel grids. Qi et al. [232] propose subvolume supervision and anisotropic kernels for 3D CNNs, and systematically analyzed



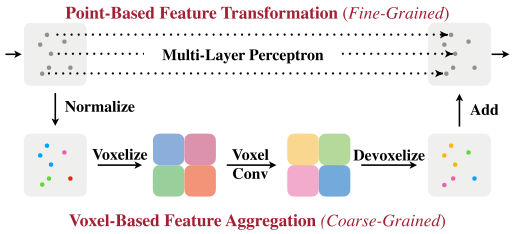
(a) Voxel-based convolution [206].



(b) Point-based convolution [173, 232, 283].



(c) Sparse convolution [56, 98, 99].



(d) Point-voxel convolution [197, 258, 279].

Fig. 11. Overview of different 3D point cloud convolutions, where (a) and (b) are conventional approaches, while (c) and (d) are emerging efficient approaches.

the relationship between 3D CNNs and multi-view CNNs. Chang et al. [36] further extend 3D CNNs to object segmentation, which is later improved by VoxSegNet [311] with dilated convolutions and squeeze-and-excitation operations. Tchapmi et al. [283] propose SEGCloud, which uses trilinear interpolation to alleviate the information loss caused by voxelization. Voxel-based methods enjoy the regular memory access pattern, thanks to the dense volumetric representation. However, the memory footprint of these methods grows cubically as the resolution grows. As a result, these methods cannot take in input with resolution higher than $64 \times 64 \times 64$, which corresponds to 40% information loss [197].

- **Point-based Convolution.** Another stream of research directly applies deep neural networks on point clouds without converting them to voxel grids [139, 158, 229–231, 233, 260, 306, 308, 324, 336] (Figure 11(b)). PointNet [231] takes advantage of the symmetric function to process the unordered point sets in 3D. Later research [233] proposed to stack PointNets hierarchically to model neighborhood information and increase model capacity. Instead of stacking PointNets as basic blocks, PointCNN [173] and SpiderCNN [330] abstract away the symmetric function using dynamically generated convolution kernels or learned neighborhood permutation function, and PointConv [320] considers point cloud density while generating dynamic kernels. By applying different kernel to different neighborhood points, these methods have better model capacity comparing with PointNet++ and usually achieve better performance. PointConv achieves better performance on indoor scenes, while PointCNN has superior accuracy on 3D objects. InterpCNN [206] and KPConv [284] propose to generate convolution kernels via interpolation. This is more efficient than learning-based dynamic kernel generation, since it does not require generating different IC×OC kernel matrix for each point. DGCNN [308] and Deep GCNs [168] model point clouds as graphs and

apply graph convolution layers to extract hierarchical features. Graph-based methods are strong in modeling small objects but usually cannot scale up to large scenes with more than 10^5 points, since the adjacency matrix alone can take up 37.3 GB of GPU memory. Point-based methods have smaller memory footprint comparing with voxel-based methods. Nevertheless, the sparsity of point cloud also brings about large irregular memory access and dynamic kernel generation cost, which takes up to 50% to 90% of total runtime [197]. Therefore, most computations are wasted on dealing with the irregularity of point cloud representation.

- **Efficient Voxel-/Point-based Convolution.** As both voxel- and point-based methods are inefficient, increased attention has been paid to the efficient design of point cloud convolution operations. To reduce the memory footprint and computation of vanilla voxel-based methods, OctNet [244] proposes to place shallow octrees within regular volumetric grids and apply convolution on this hybrid grid-octree data structure. By limiting the convolution to the octants of 3D shape boundaries instead of the interior volume, O-CNN [301] and AO-CNN [302] achieve much better speed under smaller input resolution. Other than using octrees to reduce the memory cost of volumetric CNNs, Graham et al. [98, 99] propose SparseConvNet (Figure 11(c)), which skips the non-activated regions during computation and only stores activated points. As a result, SparseConvNet can achieve orders of magnitude lower computation and memory footprint for ultra sparse, large point clouds. Choy et al. [56] further optimize SparseConvNet computation with customized matrix multiplication CUDA kernels and hashmap-based kernel map generation. The resulting MinkowskiNet is widely used in 3D instance segmentation [110, 157] and 3D detection [259, 261, 332]. It also supports high-dimensional geometric feature learning, which is applied in point cloud video understanding [56] and 3D registration [55, 57]. Another stream of research focuses on accelerating point-based methods. KPConv [284] proposes to prebuild k-d trees for input point clouds and store them on the disk. As a result, at training time, we only need to query the k-d trees on the fly instead of doing the entire distance computation and sorting as in vanilla implementations of point-based methods. This reduces the irregular data access cost during training. RandLA-Net [127] further proposes to downsample the input aggressively via random sampling to reduce computation and memory cost of point-based methods.
- **Efficient Hybrid Convolution.** Recently, there are also hybrid approaches that take advantage of the virtue of both voxel-based and point-based methods. Liu et al. [193, 197] propose Point-voxel Convolution (Figure 11(d)), which does convolution on a voxel-based branch and keeps fine-grained information on a point-based branch. The voxel-based branch does not have to maintain high resolution, thanks to the high resolution point-based branch. However, the point-based branch applies simple multi-layer perceptron to transform the high resolution features, which does not require inefficient irregular memory access operations and dynamic kernel generation. Consequently, these two branches enjoy the benefit of both regular memory access pattern and small memory footprint, which eventually leads to superior efficiency. Later research Grid-GCN [329] extends similar idea to graph neural networks and also achieves significant speedup over existing GCN-based methods. To achieve better efficiency on larger input point clouds, Tang et al. [280] propose SPVCNN, which upgrades the voxel-based branch in Point-voxel Convolution to a sparse tensor branch, and applies sparse 3D convolutions to aggregate neighborhood features.

We summarize the results of different point cloud CNNs in Tables 4, 5, and 6.

Table 4. Summarized Results of 3D Object Segmentation on the ShapeNet Dataset

	#Params (M)	#MACs (G)	GPU Latency (ms)	Mean IoU (%)
3D-UNet [58]	8.1	2996.9	682.1	84.6
PointNet [231]	2.5	10.3	21.7	83.7
PointNet++ [233]	1.8	4.9	77.9	85.1
RS-Net [129]	6.9	1.4	74.6	84.9
DGCNN [308]	1.5	18.5	87.8	85.1
SpiderCNN [330]	2.6	10.6	170.7	85.3
PointConv [320]	21.6	11.6	163.7	85.7
PointCNN [173]	8.3	26.9	135.8	86.2
KPConv [284]	14.2	–	56.8	86.4
MinkowskiNet [56]	21.7	–	–	85.1
PVCNN [197]	4.2	15.3	50.7	86.2

In this table, the latency is measured on 8 × 2,048 input points with a single NVIDIA GTX1080Ti GPU, and the number of MACs is estimated based on 2,048 input points.

Table 5. Summarized Results of 3D Indoor Scene Segmentation on S3DIS [8, 9] and ScanNet [64] Datasets

	#Params (M)	#MACs (G)	Latency (ms)	S3DIS		ScanNet	
				mAcc	mIoU	mAcc	mIoU
3D-UNet [58]	14	2,796.8	574.7	86.1	54.9	–	–
MinkowskiNet [56]	21.7	28.2	61.3	–	65.4	–	73.6
PointNet [231]	1.2	28.8	20.9	82.5	43.0	–	–
PointNet++ [233]	1.0	9.6	26.8	–	–	84.5	33.9
RS-Net [129]	6.9	17.6	111.5	–	51.9	84.9	34.9
DGCNN [308]	1.0	295.2	178.1	83.6	47.9	–	–
PointConv [320]	21.7	112.0	210.3	–	–	–	66.6
PointCNN [173]	11.5	140.0	282.3	85.9	57.3	85.1	45.8
KPConv [284]	14.1	25.6	71.3	–	67.1	–	68.4
PVCNN [197]	2.6	104.0	47.3	86.7	56.1	–	–
PVCNN++ [197]	13.7	209.6	69.5	87.1	59.0	–	–

In this table, the latency is measured with a single NVIDIA GTX1080Ti GPU. Both latency and #MACs are estimated on the full scene for MinkowskiNet and on 8 × 4,096 points or equivalent size for all other methods.

- **3D Object Part Segmentation.** We summarize the results of recent 3D deep learning methods on ShapeNet [36] in Table 4. The latency of KPConv [284] (which adopts heterogeneous batching) is estimated by projecting input size to 16,384, and the results of MinkowskiNet [56] are cited from Xie et al. [324]. Most point-based methods [129, 173, 233, 308, 320, 330] suffer from high irregular memory access and dynamic kernel computation cost and therefore run slowly on GPUs. KPConv [284] and PVCNN [197] achieve state-of-the-art accuracy while maintaining good efficiency (up to 2.7× speedup, 1.9× parameters reduction over PointCNN).
- **3D Indoor Scene Segmentation.** We summarize the results on S3DIS [8, 9] and ScanNet [64] in Table 5. The latency and #MACs of all methods except MinkowskiNet [56] are obtained on a batch of 32,768 points. We measure the scene-averaged latency and #MACs across the whole S3DIS dataset for MinkowskiNet, since it does not consume sliding window-based input as other methods. We observe that MinkowskiNet achieves best accuracy and efficiency

Table 6. Summarized Results of 3D Outdoor Scene Segmentation on the SemanticKITTI [16] Dataset

	#Params (M)	#MACs (G)	GPU Latency (ms)	Mean IoU (%)
PointNet [231]	3.0*	–	500*	14.6
SPGraph [159]	0.3*	–	5,200*	17.4
PointNet++ [233]	6.0*	–	5,900*	20.1
PVCNN [197]	2.5	42.4	146	39.0
PVCNN (sliding window) [197]	2.5	42,400	2,500	56.4
TangentConv [281]	0.4*	–	3,000*	40.9
RandLA-Net [127]	1.2	66.5	880 (256 + 624)	53.9
KPConv [284]	18.3	207.3	–	58.8
MinkowskiNet [56]	21.7	114.0	294	63.1
SPVNAS [280]	2.6	15.0	110	63.7
SPVNAS [280]	12.5	73.8	259	66.4

Here, red numbers correspond to the computation time, and blue numbers correspond to the post-processing time. *: results directly taken from Behley et al. [16].

on indoor semantic segmentation tasks, while KPConv [284] is the best among point-based methods.

- **3D Outdoor Scene Segmentation.** We summarize results of different outdoor scene segmentation methods in Table 6, where we report per-scene latency and #MACs statistics. Due to the large spatial size of outdoor LiDAR scans, point-based methods are usually very inefficient, since they have to inference on a large number of sliding windows. Methods based on sparse convolution [56, 280] have the best accuracy and efficiency among all methods. SPVNAS [280] applies neural architecture search and achieves 3× measured speedup and 8× computation reduction over MinkowskiNet [56].

Neural Architecture Search for Point Clouds. Apart from point convolution operation design, neural architecture design also plays an important role in efficient point cloud processing. Early research [56, 233, 284] takes advantage of existing image CNN designs (e.g. residual connections, U-Net structure) to manually assemble point cloud convolution layers. However, these manually designed networks are often inefficient and suboptimal. Consequently, recent research starts to focus on automated neural architecture design for efficient point cloud processing.

V-NAS [363] is tailored for volumetric representation. It leverages differentiable neural architecture search to explore a hybrid design space composed of 2D, 3D, and pseudo-3D convolution operations. However, V-NAS is targeted for medical image segmentation and cannot be directly applied to 3D point cloud processing. For point-based methods, SGAS [169] explores a DARTS [187]-like network topology with graph convolution layers as candidate operations in the search space. Built upon SGAS, LC-NAS [170] further incorporates the hardware feedback into the pipeline by training a differentiable latency regressor to predict the network latency on target platforms. The predicted latency acts as a regularization term and soft constraint during training. SGAS and LC-NAS are currently designed for 3D object classification or point cloud part segmentation and have not demonstrated their effectiveness in large-scale outdoor scene understanding.

Recently, Tang et al. [198, 281] propose 3D-NAS framework as a general tool to automatically design point cloud processing networks under resource constraints. The 3D-NAS framework supports different candidate networks with fine-grained channel numbers, elastic network depth, and resolution, allowing the #MACs of the subnets to span over a 16× range. To support such a large range of models, 3D-NAS follows a two-stage pipeline, which is similar to ProxylessNAS [29] and OFA [25]. In the first stage, a super network that contains all the subnets in the design space is

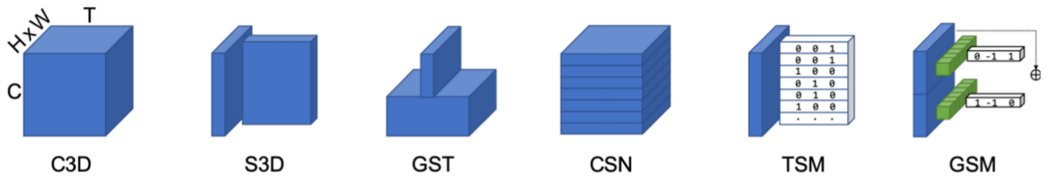


Fig. 12. Spatio-temporal decomposition of 3D convolutions for efficient video recognition [269].

trained. Thanks to the weight sharing, heterogenous sampling, and progressive depth shrinkage techniques, the subnets are able to achieve the same level of accuracy comparing with training from scratch. Subsequently, a biology-inspired evolutionary architecture search process is performed to derive the best candidate network under efficiency constraints (e.g., #MACs, latency). Different from LC-NAS, 3D-NAS enforces hard efficiency constraints, where candidates exceeding the computation budget are directly discarded. 3D-NAS can be applied on both 3D object segmentation and large-scale LiDAR scene processing, and consistently achieves significantly better performance-latency tradeoff than the manually designed ones.

5.2 Efficient Video Recognition

Efficient video understanding is an important step towards real-world deployment, both on the cloud and on the edge. For example, there are over 10^5 hours of videos uploaded to YouTube every day to be processed for recommendation and ads ranking; tera-bytes of sensitive videos in hospitals need to be processed locally on edge devices to protect privacy. All these industry applications require both accurate and efficient video understanding.

2D vs. 3D CNN. Video recognition is different from image recognition, mainly due to the temporal information. Using the **2D CNN** on each of the video frames is a straightforward way to conduct video recognition [19, 84, 85, 89, 145, 262, 300]. For example, **Temporal Segment Networks (TSN)** [300] extracted averaged features from strided sampled frames. Such methods are more efficient compared to 3D counterparts but cannot infer the temporal order or more complicated temporal relationships. **3D CNNs** can jointly learn spatio-temporal features [33, 273, 285]. For example, Tran et al. [285] proposed a 3D CNN based on VGG models, named C3D, to learn spatio-temporal features from a frame sequence. 3D CNNs usually achieve superior action recognition performance given enough data. However, 3D CNNs are computationally heavy, making the deployment difficult.

RGB vs. Flow. Using optical flow as an extra input modality can improve the video recognition accuracy [342], which is also called "two-stream" method. For example, Simonyan et al. [262] designed a two-stream CNN for RGB input (spatial stream) and optical flow [342] input (temporal stream), respectively. Despite the higher accuracy, the optical flow is expensive to compute. Therefore, it is usually not used for efficient edge video processing. A recent work also tried to distill the optical flow stream information into the RGB stream during training [267] so it does not require the temporal stream during testing.

Online vs. Offline. Most of the video understanding methods focus on offline video processing, where all the video frames are available and can be processed as a batch. On some deployment scenarios, such as smart camera, autonomous driving, and so on, it is required to perform online video understanding on a streaming video input to give low-latency feedback, usually on a per-input frame basis. TSM [177] can be used for online video recognition while still modeling temporal information. It does not incur duplicated computation, unlike Zhu et al. [362].

Using 3D convolutions for spatio-temporal modeling can be redundant [177]. To leverage the temporal redundancy and improve the efficiency, researchers have introduced efficient primitives to reduce the cost of spatio-temporal modeling by *decomposing* the spatial and temporal modeling. One method is to decompose the spatial and temporal dimension of the 3D convolutional kernels (see Figure 12). P3D [236], R(2+1)D [287], and S3D [325] decompose the 3D kernels into a 2D spatial convolution and a 1D temporal convolution, which preserves the accuracy well. Another design is to separate the channel interaction and spatio-temporal interaction with group convolutions and depthwise convolutions [286], or to decompose the feature channels into spatial and temporal groups in parallel with group convolution [201]. **Temporal Shift Module (TSM)** [177] demonstrates that the temporal modeling part can be performed with a hardware-efficient address shift without incurring extra computation. **Gate-Shift Module (GST)** [269] replaces the hard-wired channel split in TSM with a learnable spatial gating block to further improve the accuracy. Depthwise **Temporal Aggregation Module (TAM)** [80] enables the exchange of temporal information between frames by weighted channel-wise aggregation. The efficient primitives with spatial-temporal decomposition greatly improves the accuracy vs. speed/computation tradeoff. TSM models greatly outperform 3D CNNs (I3D) and an efficient mixed 2D/3D design (ECO) at the same computation per video. The efficiency of the model design also improves the training scalability significantly. Lin et al. [176] scale up the training to 1,536 GPUs, finishing the large-scale training on Kinetics [33] in one hour.

Neural Architecture Search for Videos. Neural architecture search has been employed to improve the efficiency of video understanding networks. EvaNet [228] is the first attempt to perform neural architecture search for video architectures. AssembleNet [248] uses NAS to design new methods for fusing different sub-networks with different input modalities (RGB and optical flow) and temporal resolutions. Tiny Video Networks [227] automatically design highly efficient models for video understanding. X3D [83] progressively expands a tiny 2D image classification architecture along multiple network axes, in space, time, width, and depth, to get a family of efficient video networks. Recently, Wang et al. [310] propose **Practical Video Neural Architecture Search (PV-NAS)** to efficiently search across a tremendously large scale of architectures in a novel spatial-temporal network search space using the gradient-based search methods.

5.3 Efficient Natural Language Processing

Natural Language Processing (NLP) is the key technique for numerous real-world applications, including machine translation, document summarization, and chatbots. Tradition NLP models are based on **Recurrently Neural Network (RNN)** and **Convolutional Neural Network (CNN)** [126, 148, 191, 210, 271, 349]. Lately, the NLP area is witnessing much faster advancements by virtue of the invention of the *attention mechanism* [10, 290]. Attention-based NN models such as Transformer [290], BERT [71], GPT-2 [237], GPT-3 [22], and Switch Transformer [82] provide significant performance improvements over models based on CNN and RNN. BERT [71] even outperforms human performance on the challenging question answering [239] and sentence classification [292] tasks. To pursue higher performance, the sizes of recent attention-based models are increasing exponentially. The exploding model size and computation complexity bring severe efficiency issues, making it extremely challenging to deploy NLP models on resource-limited edge devices. For instance, to translate a sentence with only 30 tokens, a Transformer-Big model needs to execute 13 G MACs and takes 20 seconds on a Raspberry Pi. Such long latency will hurt the user experience and make real-time NLP applications impossible on mobile devices. Therefore, efficient NLP techniques are of pressing demand.

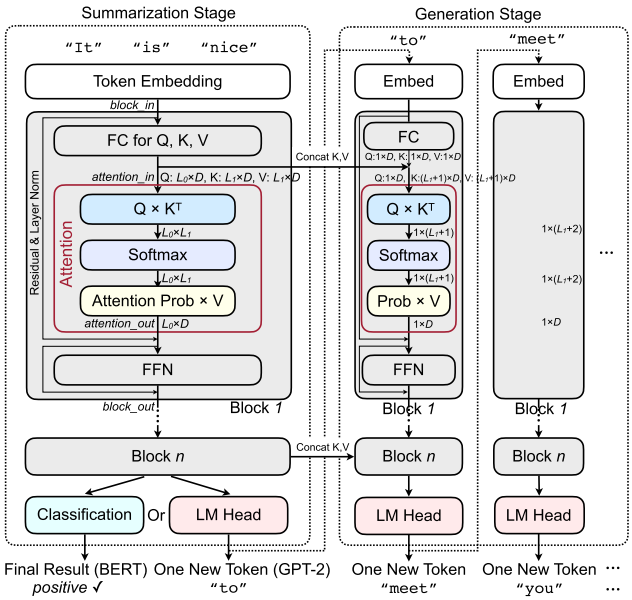


Fig. 13. Attention-based NLP model architectures [297]. BERT only contains the summarization stage, while Transformer and GPT-2 contain both summarization and generation stages.

NLP tasks can be categorized into two types: discriminative and generative. For discriminative ones, the models need to summarize the input information and make predictions. Discriminative tasks include token-level classification, sentence-level classification, and regression. Meanwhile, models for generative tasks need to summarize the input information first and then generate new tokens. Examples of generation tasks include **Language Modeling (LM)** [237], machine translation [290], and text summarization [318]. Figure 13 illustrates BERT (for discriminative tasks) and GPT-2 and Transformer (for generative tasks). BERT only contains the summarization stage, while GPT-2 and Transformer models first perform summarization and then generation stage.

In the summarization stage (Figure 13, left), the input tokens are first embedded into vectors and processed by blocks. Inside each block, `block_in` is first multiplied with three matrices to get **Query (Q), Key (K)**, and **Value (V)**. Then Q, K, and V are processed by attention to obtain the intermediate features `attention_out`. A residual layer adds the `attention_out` with `block_in` and performs layer normalization. Furthermore, a **Feed-Forward Network (FFN)** layer containing two **Fully Connected (FC)** layers is applied. Finally, another residual operation is conducted and outputs `block_out`. The same block is repeated multiple times, such as 12 times for BERT-Base. The last block is followed by one classification layer in BERT to get the *final result*. In contrast, GPT-2 applies an LM head to generate one new token and then enter the generation stage.

The generation stage (Figure 13, right) has two main differences from the summarization stage: (i) Each iteration only processes *one single token* instead of the whole sentence. (ii) Ks and Vs from the summarization stage are concatenated with current K and V and sent to attention *in batch*, while the query is still *one single vector*. After the last block, another new token will be generated. The generation stage ends when the "end of sentence" token is generated, or the sentence length reaches a pre-defined limit.

The attention mechanism is shown in Algorithm 1. In the summarization stage, K, Q, and V are matrices with the same dimension, while in the generation stage, Q is one single vector, and

ALGORITHM 1: Attention

Inputs: query $Q_{\text{in}} \in \mathbb{R}^{L_0 \times D_{\text{in}}}$, key $K_{\text{in}} \in \mathbb{R}^{L_1 \times D_{\text{in}}}$, value $V_{\text{in}} \in \mathbb{R}^{L_1 \times D_{\text{in}}}$, number of heads h

Split Q_{in}, K_{in}, V_{in} to h chunks: $Q \in \mathbb{R}^{h \times L_0 \times D}$, $K \in \mathbb{R}^{h \times L_1 \times D}$, $V \in \mathbb{R}^{h \times L_1 \times D}$, where $D = D_{\text{in}}/h$
for $i = 1$ **to** h **do**
  score $\in \mathbb{R}^{L_0 \times L_1}$
  score $= Q_i \cdot K_i^{\text{T}}/\text{sqrt}(D)$
  prob $\in \mathbb{R}^{L_0 \times L_1}$
  for $k = 1$ **to** L_0 **do**
    $\text{prob}_k = \text{softmax}(\text{score}_k)$
  end
  $O_i = \text{prob} \cdot V_i$
end
Concatenate $\{O_i \mid 1 \le i \le h\}$ as output $O \in \mathbb{R}^{h \times L_0 \times D}$

K, V are matrices. Attention has multiple *heads*, each processing a chunk of K, Q, and V. Different heads capture various dependencies between tokens, some for long-term, and some for short-term. Inside each head, $\text{Q} \times \text{K}^{\text{T}}/\text{sqrt}(D)$ gives attention scores, where D is the dimension of K, Q, and V in one head. The attention scores indicate whether two tokens are related. Intuitively, each token is looking for the most important tokens to it. After that, a row-wise softmax computes the attention probabilities. The exponential of softmax further enlarges the attention scores for highly related token pairs. The feature of the head is then computed with prob $\times$ V. This step lets each token fetch information from their cared tokens. Finally, multiple heads are concatenated together as the attention output. If there is more than one head, then an additional FC layer is applied to the attention output O.

Efficient NLP Primitive. Although the attention mechanism is the current workhorse in mainstream NLP models, research progress has been made to improve attention or replace it with new primitive operations. DeFormer [32], EdgeBert [275], Block-wise self-attention [235], and Parmar et al. [223] restrict the size of attention receptive field to reduce cost. Moreover, Reference [15] explores to use learned linear combinations of encoder outputs as the decoder inputs. Reference [303] also proposes to train deep Transformers by propagating multiple layers together in the encoder. The **Hardware-Aware Transformer (HAT)** [295] proposes arbitrary encoder-decoder attention to break the information bottleneck between the encoder and decoder by allowing decoder layers to get information from arbitrary and multiple encoder layers. Heterogeneous layer is also proposed in HAT to make different encoder and decoder layers to have various architectures, such as different numbers of heads, embedding dimensions.

Some researchers propose to replace the attention mechanism with other operations. Wu et al. [318] introduce a convolution-based module to replace the attention. The benefit of using convolution is its better local feature extraction ability. Also, the computation scales linearly instead of quadratic growth in attention. They propose lightweight depth-wise separable convolution to reduce the model size and develop dynamic convolutions built on depth-wise convolution to make the kernel adaptive to the current input context. Wu et al. [321] further this direction by proposing a hybrid **Long-Short Range Attention (LSRA)** in which one branch is convolution and another is attention. By specializing convolution and attention branch for short-range and long-range feature extractions, it achieves better accuracy-efficiency tradeoffs. Memory-compressed attention [190] employs convolution layers to shrink the sentence length. Reformer [150] replaces dot-product attention with locality-sensitive hashing to reduce computation complexity and replaces standard

residuals with reversible residual layers, which reduces peak memory consumption because the residual information dimension is reduced compared to normal residual layers. Routing Transformer [246] leverages k-means clustering of the tokens. Set Transformer [165] is proposed to model interactions among elements in the input set with an auxiliary memory. **Average Attention Network (ANN)** [345] replaces the original dynamically computed attention weights with fixed average weights. SRU [166] proposes recurrent units to replace attention. Linformer [304] approximates the attention with low-rank matrix, thus reducing attention complexity from $O(n^2)$ to $O(n)$. Reservoir Transformer [257] interleaves non-linear reservoir layers with regular attention layers. DeeBert [326] explores early exit of BERT layers to reduce computation cost. Another research stream focuses on applying non- or partially autoregressive models to cut down the iteration number for decoding [3, 102, 103, 313] by generating more than one token in one model forward pass.

Automated Compression and Neural Architecture Search for NLP. Besides new primitive operations, research efforts have been made towards automated model compression and neural architecture search for NLP models. The numerous FC layers in NLP models are promising for weight pruning and quantization. GOBO [343] proposes to compress BERT model down to 3 bits, thus significantly reducing DRAM access. Q-BERT [258] proposes a group-wise quantization with a Hessian-based mix-precision strategy. I-BERT [147] proposes integer-only BERT. TernaryBERT [348] and BinaryBERT [11] further quantize the model down to Ternary ({-1, 0, +1}) and binary schemes. Tambe et al. [276] propose an adaptive floating-point data format in consideration of the large dynamic range of NLP models' weights. Weight pruning is also widely used in NLP model size reduction [96, 174, 333]. The major differences between them are the number of bits and the granularity of quantization. According to Bai et al. [11], on different tasks, the BERT accuracy goes down from 3% to 7% when quantized to 4 bits, and another 4% to 13% when quantized to 2 bits.

Besides, NLP models have large opportunities for activation pruning and quantization because of the redundancy of human languages. Multiple on-the-fly pruning methods are proposed to reduce the redundancy by token/head pruning [97, 146, 209, 291, 293, 297]. Specifically, SpAtten [297] proposes *cascade token/head pruning* to reduce both DRAM access and computation. It prunes the tokens according to cumulative token importance scores obtained by accumulating attention probabilities (indicators for token influence) across layers. The heads are pruned based on cumulative head importance scores, which are computed by accumulating each head's magnitude across layers. Longformer [17] and Sparse Transformer [53] also leverage activation pruning by sparsifying the attention connections—only attending to one token from every several tokens. For quantization, SpAtten [297] proposes *progressive quantization* for attention inputs. It quantizes more aggressively for attention with dominated attention probabilities and more conservatively for others. Concretely, it first fetches MSBs of attention inputs to compute the attention probabilities. If the maximum probability is smaller than a threshold, indicating the distribution is flat, then it will fetch LSBs on-chip and recompute attention probabilities.

Besides pruning and quantization, research efforts have also been invested on the automatic search for efficient NLP model architecture. Considering different properties of various hardware platforms [60], **Hardware-Aware Transformer (HAT)** [295] is proposed. It finds that FLOPs cannot reflect the measured latency, and different hardware prefers different Transformer architecture. Therefore, a specialized NLP model for each hardware is necessary. Specifically, it first constructs a large design space with arbitrary encoder-decoder attention and heterogeneous layers. Then HAT trains a SuperTransformer that covers all candidates in the design space and efficiently produces many SubTransformers with weight sharing. Finally, it performs an evolutionary search with a

Table 7. Summarized Results of Transformer [290], Evolved Transformer [264], and HAT [299]

		Latency (s)	#Params (M)	FLOPs (G)	BLEU	GPU Hours	CO_2e (lbs)	Cloud Comp. Cost
IWSLT'14 De-En	Transformer [290]	3.3	32	1.5	34.5	2	5	$12–$40
	HAT [299]	2.1	23	1.1	34.5	4	9	$24–$80
WMT'14 En-Fr	Transformer [290]	23.2	176	10.6	41.2	240	68	$178–$595
	ET [264]	20.9	175	10.8	41.3	2,192,000	626,000	$1.6M–$5.5M
	HAT [299]	7.8	48	3.4	41.4	216	61	$159–$534
	HAT [299]	9.1	57	3.9	41.8	224	64	$166–$555
WMT'14 En-De	Transformer [290]	20.5	176	10.6	28.4	184	52	$136–$456
	ET [264]	7.6	47	2.9	28.2	2,192,000	626,000	$1.6M–$5.5M
	HAT [299]	6.0	44	2.7	28.2	184	52	$136–$456
	HAT [299]	6.9	48	3.0	28.4	200	57	$147–$495

In this table, the latency is measured on the Raspberry Pi ARM CPU, the training cost is estimated with a single NVIDIA V100 GPU, and the CO_2 emission and the cloud computing cost are computed following Strubell et al. [268].

hardware latency constraint to find a specialized SubTransformer dedicated to run fast on the target hardware. Table 7 shows the performance comparison between HAT and state-of-the-art models. Similarly, AutoADR [47] also searches for a model under certain hardware constraints (memory, latency). AdaBert [38] leverages differentiable supernet-based NAS and knowledge distillation to automatically compress BERT into task-adaptive small models for specific tasks. AutoEmb [353] searches for the embedding size according to the input popularity. AttentionNAS [361] designs a large design space containing various operations to search for an efficient attention cell.

5.4 Discussions and Future Directions

In this section, we introduced efficient approaches for emerging applications such as 3D point cloud processing, video understanding, and natural language processing. For efficient 3D deep learning, we believe there will be new research on hardware-aware automated point cloud network design for different tasks, such as 3D object detection and panoptic segmentation. System-algorithm codesign will be another important topic: current state-of-the-art implementations of 3D deep learning modules only consider optimizing single layer or single operation, which is usually model-agnostic and has large room for improvement. For efficient video understanding, new primitive design and automated architecture search will still play an important role in future research. Since video annotation is usually more expensive than image labeling, data efficient video understanding will also receive attention in the future. For efficient natural language processing, one future research direction can be One-For-All Transformer, in which we can train one super transformer and deploy it to various edge devices without large training cost. Since most of the current state-of-the-art models require pre-training on a large corpus, another future research direction can be increasing the efficiency of the pre-training stage such as incorporating sparsity and quantization. That may help reduce the prohibitive bar for model pre-training and cultivate more efficient backbone model architectures. Since the auto-regressive GPT-2 model spends most of the runtime in the generation stage, combining existing partial or non-autoregressive model architectures with attention operation has the potential to significantly reduce the latency. Finally, algorithm-hardware co-design across the stack is another promising methodology to address the challenge of efficient NLP.

6 EFFICIENT SYSTEM DESIGN

Approaches introduced in the previous sections are top-down solutions for efficient deep learning processing, where performance gains come from the algorithmic optimization. However, these *theoretical* benefits (e.g., FLOPs and model size) cannot be easily converted into the *real* improvements in measured speedup and energy efficiency. Therefore, specialized software/hardware systems are

required to bridge the gap. However, these specialized software/hardware systems open up a new design space orthogonal to the algorithm space, bringing the opportunities for holistic optimization by jointly optimizing the algorithm and software/hardware systems.

6.1 Software System

Deep learning software systems include general training & inference libraries (PyTorch [224], TensorFlow [1], and MXNet [42]), hardware-acceleration libraries (cuDNN [52] and MKLDNN), and model-serving libraries (TVM [43] and MNN [141]).

Researchers have been developing specialized deep learning systems, some with *learning-based* methods for optimization. TVM [43] is a compiler that for deep learning workloads across diverse hardware backends. It automates optimization of low-level programs to hardware characteristics with a learning-based method. FlexFlow [138] introduces functional-preserving graph transformations to optimize DNN executions. TASO [137] further introduces an automated generation of substitution rules. MetaFlow and TASO can take the whole graph into consideration and search for high optimized substitution strategies and achieve up to 3× speedup over existing DNN frameworks. For high-end server accelerators, IOS [72] explores the inter operator parallelism beyond intra operator ones, as a single operator can no longer fully utilize the accelerator because of hardware advances. For low-end micro-controllers, TinyEngine [175] generates model-specific memory scheduling for deep learning inference on memory-constrained microcontrollers.

Recently, there are also emerging domain-specific software systems. For example, in point cloud processing, Kaolin [153] collects implementations of state-of-the-art point cloud operators. PyTorch3D [242] features both point-based 3D module acceleration and efficient differentiable rendering. PyTorch3D accelerates implementation for D-dimensional k-NN with specialized tuning for different (D, k) pairs. Besides, graph convolution operations are improved in memory efficiency in PyTorch3D through fusing `gather+scatter_add` within a single CUDA kernel. SpConv [332], MinkowskiEngine [56], and TorchSparse [280] are specialized for sparse tensor operations (especially sparse convolution) on 3D point clouds. These libraries differ in implementations of kernel map construction (i.e. convolution rule generation) and convolution. SpConv implements kernel map construction with volumetric binary tensor lookup, and MinkowskiEngine features CPU parallel hashtable-based solution. TorchSparse further optimizes the efficiency of kernel map construction with a GPU hashmap and parallel zero eliminator. For convolution implementation, MinkowskiEngine features direct computation while SpConv prefers the `Gather-MatMul-Scatter` dataflow. TorchSparse improves the memory efficiency of `Gather-MatMul-Scatter` dataflow while also providing support for direct computation of convolution.

6.2 Hardware System

For most CPUs and GPUs, data for ALU can only be fetched from the memory hierarchy and cannot communicate directly with each other; thus, the common approach to accelerate neural networks is vectorization (SIMD) and multi-threading (SIMT). Nonetheless, domain-specific accelerators are able to obtain additional performance and efficiency gain via four main techniques [66]: data specialization, parallelism, local and optimized memory, and reduced overhead. Therefore, designing specialized hardware system is a popular bottom-up way for efficient deep learning processing.

General Deep Learning Accelerators. General deep learning accelerators are mostly spatial architectures using dataflow processing [272] where ALUs form a processing chain so they can pass data from one to another directly. Since the area, power, as well as performance of most accelerators are dominated by memory, data handling characteristics, i.e., dataflow, can be used to

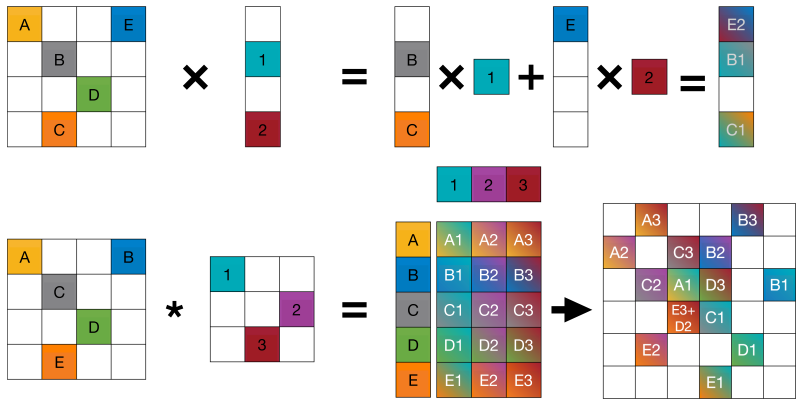


Fig. 14. Upper: Sparse matrix-vector multiplication using compressed sparse column (CSC) format in EIE [113]. Lower: Sparse convolution using Cartesian product in SCNN [219].

classify the deep learning accelerators. Weight stationary dataflow is widely used [34, 35, 221, 252, 266], where the weights are stored in the local memory and reused as much as possible. On the contrary, output stationary dataflow [75, 107, 225] maximizes the reuse of output partial sums that are accumulated locally. Instead of maximizing the reuse of single type of data, row stationary dataflow [48, 49, 90, 107] assigns 1-D row convolution to each PE to maximize the reuse of all data types (inputs, weights, and outputs). Besides, no-local-reuse dataflow [41, 45, 346] increases the global buffer capacity and minimizes the off-chip memory bandwidth by eliminating the local memory. Lu et al. [200] also present an NLP accelerator with support of efficient matrix partition and on-chip PE reuse. Many researchers also leverage specialized operators or devices for NLP acceleration. ATT [104] proposes a ReRAM-based architecture to support attention operations and adds non-uniform redundancy to improve accuracy. Park et al. [222] propose an FPGA-based accelerator for the Question Answering task. RAMANN [2] designs a specialized 9T SRAM cell for efficient dot production in memory-augmented networks.

Hardware Support for Sparsity. Several works [220, 309] propose to leverage existing general-purpose CPUs/GPUs to support sparsity in pruned NN models. Others propose domain-specific hardware accelerators that bring much high efficiency at the cost of longer design cycle and larger design automation burden [205, 294, 296]. Compressing the sparse data is a straightforward way to save energy by reducing the data transfer cost. An example is Eyeriss [48], which uses a run length encoding scheme to compress activations being transferred to/from DRAM and gates the multiplier for zero activations.

Skipping the multiplication whose operand is zero is another natural way to save time. EIE [113] accelerates the sparse matrix-vector multiplication specifically for the fully connected layers. It stores the weights in a CSC format along with the start location of each column. When the input activation is not zero, the compressed weight column is read and the corresponding output is updated as shown in Figure 14. For sparse convolution acceleration, Cnvlutin [4] similarly selects weights for multiplication based on only non-zero input activations, while Cambricon-X [347] contrarily selects the input activations based on the non-zero weights. SCNN [219] further supports convolution in a compressed format for both input activations and weights. It uses an input stationary dataflow to deliver the compressed weights and activations to a multiplier array and performs Cartesian product in parallel to compute the partial sums followed by a scatter network to accumulate these scattered partial sums, as shown in Figure 14.

Generalized Sparse Matrix-Matrix Multiplication (SpGEMM) accelerators can also be used for sparse deep learning inference [218, 234, 352]. SpArch [352] is a specialized accelerator for sparse-sparse matrix multiplication. It jointly optimizes the input and output matrix data reuse. SpArch first designs a highly parallelized *streaming-based merger* to pipeline the multiplication and merge stage of partial matrices so partial matrices are merged on chip immediately after produced, and then uses a *condensed matrix representation* to reduce the number of partial matrices by three orders of magnitude. SpArch further applies a *Huffman tree scheduler* to improve the scalability of the merger for larger sparse matrices and uses a *row prefetcher* with near-optimal buffer replacement policy to deal with the increased input matrix read. A number of designs are also presented to accelerate sparse linear algebra on FPGA platforms. Zhuo et al. [364] introduce a tree of binary operators to increase the energy efficiency of SpMV on FPGAs. Jamro et al. [135] prove that separating indices comparison and computing operations could increase the throughput. The index comparison and computing in the SpAtten system design are also separated. Zou et al. [367] propose a new sparse matrix storage method called "BVCSR" to compress the indices of non-zero elements, thus increasing the valid bandwidth of FPGA. Elkurdi et al. [76] and Grigoracs et al. [100] propose an architecture for large-scale SpMV in the FEM problem. Elkurdi et al. [76] co-design an FPGA SpMV architecture with a matrix stripping and partitioning algorithms that enable the architecture to process arbitrarily large matrices without changing the PE quantities.

Apart from sparsity in weights and activations, some accelerators also explore the sparsity of query, key, and value vectors in the attention layer. Pruning attention is fundamentally different from weight pruning because, as introduced in Section 5.3, there are no weights in the attention part. A^3 [108] first sorts each dimension of the key vectors among all keys. Then it uses a prespecified number of largest/smallest elements in the keys to conduct multiplications with a query and get *partial* attention scores. The corresponding key will be pruned if a score is smaller than a threshold. MNNFast [136] removes V vectors whose attention probabilities are smaller than a threshold. SpAtten [297] proposes an accelerator architecture to support attention computation with specialized high-parallelism top-k engine for token/head selections, specialized memory hierarchy, and fully pipelined datapath to translate theoretical savings to real speedup and energy reduction.

Hardware Support for Quantization. Quantized models can reduce the model size and storage for deployment, but it requires hardware support on low-precision arithmetic for inference acceleration. INT8 quantization is supported on mobile ARM CPUs (e.g., Qualcomm Hexagon, ARM Neon), x86 CPUs, NVIDIA GPUs with TensorRT, and Xilinx FPGAs with DNNDK. NVIDIA's Turing architecture is able to further support INT4 inference, which brings an additional 59% speedup compared to INT8. Since lower bit widths are less supported on existing hardware, specialized hardware implementations for binary and ternary neural networks have also been studied [6, 7, 288]. To achieve better accuracy vs. cost tradeoff, hardware support for *mixed-precision* quantization has been extensively explored. NVIDIA's Turing Tensor Core supports 1-bit, 4-bit, 8-bit, and 16-bit arithmetic operations; Imagination launched a flexible neural network IP that supports per-layer bit-width adjustment for both weights and activations. Stripes [143] and UNPU [164] use bit-serial computation to support one mixed-precision operand (either inputs or weights). BISMO [289] and Loom [255] further adopt temporal fusion to provide full bit flexibility. Deeprecon [249] and BitFusion [256] dynamically compose and decompose 2-bit multipliers in space to construct 2/4/8-bit multiply-add units. SpAtten [297] develops hardware support for the progressive quantization technique to accelerate the memory-bounded NLP models (e.g., GPT-2).

ML-based Design and Optimization of Hardware. The surge in the demand for the computational resources for training and inferring the NN models leads to the wide usage of heterogeneous

Table 8. Correlation between Neural and Hardware Architecture Design Space [179]

Hardware Architecture Design Space	Neural Architecture Design Space					
	Input Channels	Output Channels	Kernel Size	Feature Map Size	Input Bits	Weight Bits
Array #rows	B		E		B	B
Array #cols		B		E		
IBUF Size	B/E			B/E	B/E	
WBUF Size	B/E	B/E	B/E		B	B/E
OBUF Size		B/E		B/E		

This relationship differs from hardware to hardware (B: BitFusion [256] and E: Eyeriss [48]).

environment with a combination of many CPUs, GPUs, and even FPGAs. The operations in NN are explicitly and manually placed onto the particular computing devices for model parallelism and data parallelism. Such approach does not scale well or produce the optimal results, as neural networks become more and more complicated. Recent works exploit deep reinforcement learning to automatically optimize the hardware resource assignment for training and inference of neural networks given the growing diversity of hardware devices.

A simplest setting is device placement optimization problem, which tries to map given neural networks to given hardware devices. Mirhoseini et al. [212] use a sequence-to-sequence model to process the sequence of operations and output the placement for each operation. The execution time of each proposal is applied as a reward signal to train the proposal network. Mirhoseini et al. [211] then propose an end-to-end solution with a two-level hierarchical model for proposal: The first model groups the operations in the compute graph, and the second model places these groups onto devices, getting rid of manually group operations as a pre-processing step.

Jiang et al. [140] combine the FPGA device placement with neural architecture design by conducting device placement optimization and NAS iteratively: The best pipelined FPGA configuration is identified for the proposed neural architecture candidates, and then the superior network candidates are trained for controller update using policy gradient reinforcement learning. Kao et al. [144] further try to allocate the compute resource (such as #PEs, buffer sizes) even inside single accelerator accordingly: Apart from using reinforcement learning to propose a rough assignment, genetic algorithm is applied afterwards to fine-tune the proposal.

These works only focus on using machine learning algorithms to allocate the hardware resource given the fixed hardware design, and the freedom in the hardware design, such as accelerator architecture and design parameters, is neglected.

Meanwhile, existing accelerators mostly target common neural architectures and do not reap the power of NAS. The design space of hardware and neural architectures deeply entangle with each other, as illustrated in Table 8, where the correlations are complicated and vary from hardware to hardware. For instance, tiled convolution kernels should fit into the on-chip weight buffers in both accelerators, and the product of input channels and weight bitwidth should be multiples of the compute array #rows in BitFusion. It is important to co-design the neural and hardware architectures by considering all the correlations and make them fit. Perfectly matched neural architecture and hardware architecture improve the utilization of the compute array and on-chip memory, maximizing efficiency and performance.

A straightforward approach is to integrate NAS with designing hardware accelerator. NASAIC [334] narrows down the hardware design space to the selection of limited accelerator templates and exploits the meta-controller similar to vanilla NAS [365] to predict the parameters of hardware resource allocation for different hardware template selections, achieving 17.77%, 2.49×, and 2.32× reduction in latency, energy, and area with less than 1.6% accuracy loss on CIFAR-scale datasets. NHAS [179] further expands the neural architecture design space to allow

mixed-precision quantization and expands the hardware architecture design space with hardware architectural sizing parameters, including compute array size, input/weight/output buffer sizes, and global buffer sizes. NHAS exploits the evolution algorithms to improve the sample efficiency during search, achieving 1.92×, 1.79× speedup, and 1.63×, 1.49× energy savings without hurting the accuracy of ResNet-18 and ResNet-50, respectively.

These works mainly focus on sizing the architectural hyperparameters while they neglect searching the PE connectivities and compiler mappings. **Neural Accelerator Architecture Search (NAAS)** [181] pushes beyond architectural sizing and searches PE connectivities and compiler mapping at the same time. NAAS models the PE connectivity as the choices of parallel dimensions in the computation loop nests and proposes "importance-based" encoding method to encode *non-numerical* parameters, such as indexing of parallelism in hardware optimization and ordering of for-loops in mapping optimization, into *numerical* parameters. Furthermore, NAAS integrates the Once-For-All NAS algorithm in the optimization loop. NAAS achieves 4.88× energy-delay-product improvement in total as well as 2.7% top-1 accuracy improvement on ImageNet dataset than Eyeriss running ResNet50 with a new hardware architecture design.

6.2.1 Discussions and Future Directions. Designing deep learning hardware systems requires expertise of both algorithm and hardware. Thus, exploiting machine learning to design and optimize deep learning hardware is a heated topic. However, current machine-learning-based hardware architecture design frameworks focus on sizing the architectural hyperparameters while they neglect either the lower-level design such as PE connectivities or the mapping strategies compiled by software system. Therefore, an important future research direction is to free more design freedom in the hardware architecture design and jointly optimize hardware and software systems. Current works also heavily depend on the results of simulation instead of results after a complete hardware design cycle, including synthesis and layout due to time limitation. Thus, another future research direction is to apply machine learning on each possible step in the hardware design cycle to automate the whole procedure and provide a better hardware solution than that guided by heuristic rules.

7 CONCLUSION

Over the past few years, deep neural networks have achieved unprecedented success in the field of artificial intelligence; however, their superior performance comes at the cost of high computational complexity. This limits their applications on many edge devices, where the hardware resources are tightly constrained by the form factor, battery, and heat dissipation.

In this article, we offer a systematic overview of efficient deep learning to enable both researchers and practitioners to quickly get started in this field. We first introduce various model compression approaches that have become the industry standards, such as pruning, factorization, quantization, and efficient model design. To reduce the design cost of these handcrafted solutions, we then describe many recent efforts on neural architecture search, automated pruning, and quantization, which can outperform the manual design with minimal human efforts. Apart from inference, we also cover the efficient on-device training to enable user customization based on the local data. We showcase several task-specific accelerations for point cloud, video, and natural language processing by leveraging their spatial sparsity and temporal/token redundancy. Finally, we introduce the efficient software/hardware system to support all these algorithmic improvements.

Efficient deep learning is the key enabler for many real-world AI applications. As for the future direction, it is critical to explore the extreme of efficient deep learning, ranging from cloud AI to mobile and tiny AI. To achieve this goal, we will have to accelerate deep learning from all possible perspectives, from training to inference, and from software to hardware. Furthermore, it

is promising to study the co-design of algorithm, software and hardware system, and exploit the unique properties of different domains for optimization.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*.

[2] Mohsen Ahmadzadeh, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2021. A2P-MANN: Adaptive attention inference hops pruned memory-augmented neural networks. *arXiv preprint arXiv:2101.09693* (2021).

[3] Nader Akoury, Kalpesh Krishna, and Mohit Iyyer. 2019. Syntactically supervised transformers for faster neural machine translation. In *Conference of the Association for Computational Linguistics*.

[4] Jorge Albericio, Patrick Judd, Tayler H. Hetherington, Tor M. Aamodt, Natalie D. Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *International Symposium on Computer Architecture*.

[5] Alexander Amini, Guy Rosman, Sertac Karaman, and Daniela Rus. 2019. Variational end-to-end navigation and localization. In *IEEE International Conference on Robotics and Automation*.

[6] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Masayuki Ikebe, Tetsuya Asai, Shinya Takamaeda-Yamazaki, Tadahiro Kuroda, and Masato Motomur. 2017. BRein memory: A 13-layer 4.2 K Neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm CMOS. In *Symposium on VLSI Circuits*.

[7] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2016. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. In *IEEE Computer Society Annual Symposium on VLSI*.

[8] Iro Armeni, Alexandar Sax, Amir R. Zamir, and Silvio Savarese. 2017. Joint 2D-3D-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105* (2017).

[9] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 2016. 3D semantic parsing of large-scale indoor spaces. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

[11] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2020. BinaryBERT: Pushing the limit of BERT quantization. In *Conference of the Association for Computational Linguistics*.

[12] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*.

[13] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakkar, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. 2020. MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers. In *Conference on Machine Learning and Systems*.

[14] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Conference on Neural Information Processing Systems*.

[15] Ankur Bapna, Mia Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. In *Conference on Empirical Methods in Natural Language Processing*.

[16] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. 2019. SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences. In *International Conference on Computer Vision*.

[17] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

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

[19] Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. 2016. Dynamic image networks for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[20] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[21] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2018. SMASH: One-shot model architecture search through HyperNetworks. In *International Conference on Learning Representations*.

[22] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger,

Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Conference on Neural Information Processing Systems*.

[23] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *International Conference on Knowledge Discovery and Data Mining*.

[24] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence*.

[25] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

[26] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce memory, not parameters for efficient on-device learning. In *Conference on Neural Information Processing Systems*.

[27] Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Kuan Wang, Tianzhe Wang, Ligeng Zhu, and Song Han. 2019. AutoML for architecting efficient and specialized neural networks. *IEEE Micro* 40, 1 (2019), 75–82.

[28] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. 2018. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*.

[29] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*.

[30] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. ZeroQ: A novel zero shot quantization framework. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[31] Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210* (2018).

[32] Qingqing Cao, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. DeFormer: Decomposing pre-trained transformers for faster question answering. In *Conference of the Association for Computational Linguistics*.

[33] Joao Carreira and Andrew Zisserman. 2017. Quo vadis, action recognition? A new model and the kinetics dataset. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[34] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. 2015. Origami: A convolutional network accelerator. In *Great Lakes Symposium on VLSI*.

[35] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. 2010. A dynamically configurable coprocessor for convolutional neural networks. In *International Symposium on Computer Architecture*.

[36] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012* (2015).

[37] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*.

[38] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020. AdaBERT: Task-adaptive BERT compression with differentiable neural architecture search. In *International Joint Conference on Artificial Intelligence*.

[39] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. 2017. Learning efficient object detection models with knowledge distillation. In *Conference on Neural Information Processing Systems*.

[40] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. 2018. Searching for efficient multi-scale architectures for dense image prediction. In *Conference on Neural Information Processing Systems*.

[41] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[42] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[43] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *USENIX Symposium on Operating Systems Design and Implementation*.

[44] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[45] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A machine-learning supercomputer. In *International Symposium on Microarchitecture*.

[46] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. 2019. DetNAS: Backbone search for object detection. In *Conference on Neural Information Processing Systems*.

[47] Yiren Chen, Yaming Yang, Hong Sun, Yujing Wang, Yu Xu, Wei Shen, Rong Zhou, Yunhai Tong, Jing Bai, and Ruofei Zhang. 2020. AutoADR: Automatic model design for ad relevance. In *International Conference on Information & Knowledge Management*.

[48] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *Int. J. Space-based Situat. Comput.* 52, 1 (2017), 127–138.

[49] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Select. Topics Circ. Syst.* 9, 2 (2019), 292–308.

[50] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *IEEE Sig. Process. Mag.* 35, 1 (2017), 126–136.

[51] Robin Cheong and Robel Daniel. 2019. *transformers.zip: Compressing Transformers with Pruning and Quantization.* Technical Report. Stanford University, Stanford, CA.

[52] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[53] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).

[54] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artif. Intell. Rev.* (2020).

[55] Christopher Choy, Wei Dong, and Vladlen Koltun. 2020. Deep global registration. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[56] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 2019. 4D spatio-temporal ConvNets: Minkowski convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[57] Christopher Choy, Jaesik Park, and Vladlen Koltun. 2019. Fully convolutional geometric features. In *International Conference on Computer Vision*.

[58] Ozgun Cicek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 2016. 3D U-Net: Learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*.

[59] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. 2018. End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation*.

[60] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. 2018. Understanding performance differences of FPGAs and GPUs. In *IEEE Symposium on Field-programmable Custom Computing Machines*.

[61] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Conference on Neural Information Processing Systems*.

[62] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv preprint arXiv:1602.02830* (2016).

[63] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. 2018. Large scale fine-grained categorization and domain-specific transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[64] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[65] Pengcheng Dai, Jianlei Yang, Xucheng Ye, Xingzhou Cheng, Junyu Luo, Linghao Song, Yiran Chen, and Weisheng Zhao. 2020. SparseTrain: Exploiting dataflow sparsity for efficient convolutional neural networks training. In *Design Automation Conference*.

[66] William J. Dally, Yatish Turakhia, and Song Han. 2020. Domain-specific hardware accelerators. *Commun. ACM* 63, 7 (2020).

[67] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[68] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* (2020).

[69] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, Yifan Gong, and Alex Acero. 2013. Recent advances in deep learning for speech research at Microsoft. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

[70] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Conference on Neural Information Processing Systems*.

[71] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics*.
[72] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2020. IOS: Inter-operator scheduler for CNN acceleration. *arXiv preprint arXiv:2011.01302* (2020).
[73] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. DeCAF: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*.
[74] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian AWare quantization of neural networks with mixed-precision. In *International Conference on Computer Vision*.
[75] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *International Symposium on Computer Architecture*.
[76] Yousef Elkurdi, David Fernández, Evgueni Souleimanov, Dennis Giannacopoulos, and Warren J. Gross. 2008. FPGA architecture and implementation of sparse matrix-vector multiplication for the finite element method. *Comput. Phys. Commun.* 178, 8 (2008), 558–570.
[77] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Efficient multi-objective neural architecture search via Lamarckian evolution. In *International Conference on Learning Representations*.
[78] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *J. Mach. Learn. Res.* 20, 55 (2019), 1–21.
[79] Andries Petrus Engelbrecht. 2001. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Trans. Neural Netw.* 12, 6 (2001), 1386–1399.
[80] Quanfu Fan, Chun-Fu Chen, Hilde Kuehne, Marco Pistoia, and David Cox. 2019. More is less: Learning efficient video representations by big-little network and depthwise temporal aggregation. In *Conference on Neural Information Processing Systems*.
[81] Igor Fedorov, Ryan P. Adams, Matthew Mattina, and Paul N. Whatmough. 2019. SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers. In *Conference on Neural Information Processing Systems*.
[82] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961* (2021).
[83] Christoph Feichtenhofer. 2020. X3D: Expanding architectures for efficient video recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[84] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. 2016. Spatiotemporal residual networks for video action recognition. In *Conference on Neural Information Processing Systems*.
[85] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. 2016. Convolutional two-stream network fusion for video action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[86] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.
[87] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*.
[88] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. 2020. Training BatchNorm and only BatchNorm: On the expressive power of random features in CNNs. *arXiv preprint arXiv:2003.00152* (2020).
[89] Chuang Gan, Naiyan Wang, Yi Yang, Dit-Yan Yeung, and Alex G. Hauptmann. 2015. DevNet: A deep event network for multimedia event detection and evidence recounting. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[90] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.
[91] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. 2019. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[92] C. Lee Giles and Christian W. Omlin. 1994. Pruning recurrent neural networks for improved generalization performance. *IEEE Trans. Neural Netw.* 5, 5 (1994), 848–851.
[93] Ross Girshick. 2015. Fast R-CNN. In *International Conference on Computer Vision*.
[94] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations*. Stanford University.
[95] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).
[96] Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307* (2020).
[97] Saurabh Goyal et al. 2020. PoWER-BERT: Accelerating BERT inference for classification tasks. In *International Conference on Machine Learning*.

[98] Benjamin Graham. 2015. Sparse 3D convolutional neural networks. In *British Machine Vision Conference*.
[99] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 2018. 3D semantic segmentation with submanifold sparse convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[100] Paul Grigoraş, Pavel Burovskiy, Wayne Luk, and Spencer Sherwin. 2016. Optimising sparse matrix vector multiplication for large scale FEM problems on FPGA. In *International Conference on Field-programmable Logic and Applications*.
[101] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. 2016. Memory-efficient backpropagation through time. In *Conference on Neural Information Processing Systems*.
[102] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.
[103] Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein transformer. In *Conference on Neural Information Processing Systems*.
[104] Haoqiang Guo, Lu Peng, Jian Zhang, Qing Chen, and Travis D. LeCompte. 2020. ATT: A fault-tolerant ReRAM accelerator for attention-based neural networks. In *International Conference on Computer Design*.
[105] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Conference on Neural Information Processing Systems*.
[106] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*.
[107] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*.
[108] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W. Lee, and Deog-Kyoon Jeong. 2020. A3: Accelerating attention mechanisms in neural networks with approximation. In *IEEE International Symposium on High-performance Computer Architecture*.
[109] Dongyoon Han, Jiwhan Kim, and Junmo Kim. 2017. Deep pyramidal residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[110] Lei Han, Tian Zheng, Lan Xu, and Lu Fang. 2020. OccuSeg: Occupancy-aware 3D instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[111] Song Han. 2017. *Efficient Methods and Hardware for Deep Learning*. Ph.D. Dissertation. Stanford University.
[112] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William J. Dally. 2017. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[113] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *International Symposium on Computer Architecture*.
[114] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations*.
[115] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. In *Conference on Neural Information Processing Systems*.
[116] Albert Haque, Michelle Guo, Alexandre Alahi, Serena Yeung, Zelun Luo, Alisha Rege, Jeffrey Jopling, Lance Downing, William Beninati, Amit Singh, Terry Platchek, Arnold Milstein, and Li Fei-Fei. 2017. Towards vision-based smart hospitals: A system for tracking and monitoring hand hygiene compliance. In *Machine Learning for Healthcare Conference*.
[117] Babak Hassibi and David G. Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Conference on Neural Information Processing Systems*.
[118] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[119] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowl.-based Syst.* 212 (2021), 106622.
[120] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision*.
[121] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision*.
[122] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Sig. Process. Mag.* 29, 6 (2012), 82–97.
[123] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[124] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. In *International Conference on Computer Vision*.

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

[126] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Conference on Neural Information Processing Systems*.

[127] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. 2020. RandLA-Net: Efficient semantic segmentation of large-scale point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[128] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*.

[129] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. 2018. Recurrent slice networks for 3D segmentation on point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[130] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[131] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. AI benchmark: Running deep neural networks on Android smartphones. *arXiv preprint arXiv:1810.01109* (2018).

[132] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*.

[133] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[134] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference*.

[135] Ernest Jamro, Tomasz Pabiś, Paweł Russek, and Kazimierz Wiatr. 2015. The algorithms for FPGA implementation of sparse matrices multiplication. *Comput. Inform.* 33, 3 (2015), 667–684.

[136] Hanhwi Jang, Joonsung Kim, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. 2019. MnnFast: A fast and scalable system architecture for memory-augmented neural networks. In *International Symposium on Computer Architecture*.

[137] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In *ACM Symposium on Operating Systems Principles*.

[138] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2018. Beyond data and model parallelism for deep neural networks. In *International Conference on Machine Learning*.

[139] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. 2020. PointGroup: Dual-set point grouping for 3D instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[140] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/software co-exploration of neural architectures. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 39, 12 (2020), 4805–4815.

[141] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lv, and Zhihua Wu. 2020. MNN: A universal and efficient inference engine. In *Conference on Machine Learning and Systems*.

[142] Arthur Jochems, Timo M. Deist, Issam El Naqa, Marc Kessler, Chuck Mayo, Jackson Reeves, Shruti Jolly, Martha Matuszak, Randall Ten Haken, Johan van Soest, Cary Oberije, Corinne Faivre-Finn, Gareth Price, Dirk de Ruysscher, Philippe Lambin, and Andre Dekker. 2017. Developing and validating a survival prediction model for NSCLC patients through distributed learning across 3 countries. *Int. J. Radiat. Oncol., Biol. Phys.* 99, 2 (2017), 344–352.

[143] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *IEEE/ACM International Symposium on Microarchitecture*.

[144] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning. In *IEEE/ACM International Symposium on Microarchitecture*.

[145] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[146] Gyuwan Kim and Kyunghyun Cho. 2020. Length-adaptive transformer: Train once with length drop, use anytime with search. *arXiv preprint arXiv:2010.07003* (2020).

[147] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. I-BERT: Integer-only BERT quantization. *arXiv preprint arXiv:2101.01321* (2021).

[148] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing*.
[149] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530* (2015).
[150] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2019. Reformer: The efficient transformer. In *International Conference on Learning Representations*.
[151] Jakub Konečnỳ, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).
[152] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. 2019. Do better ImageNet models transfer better? In *IEEE Conference on Computer Vision and Pattern Recognition*.
[153] Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, Sanja Fidler, Krishna Murthy Jatavallabhula, and Edward Smith. 2019. Kaolin: A Pytorch library for accelerating 3D deep learning research. *arXiv preprint arXiv:1911.05063* (2019).
[154] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto.
[155] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems*.
[156] Jason Kuen, Xiangfei Kong, Zhe Lin, Gang Wang, Jianxiong Yin, Simon See, and Yap-Peng Tan. 2018. Stochastic downsampling for cost-adjustable inference and improved regularization in convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[157] Jean Lahoud, Bernard Ghanem, Marc Pollefeys, and Martin R. Oswald. 2019. 3D instance segmentation via multi-task metric learning. In *International Conference on Computer Vision*.
[158] Shiyi Lan, Ruichi Yu, Gang Yu, and Larry S. Davis. 2019. Modeling local geometric structure of 3D point clouds using geo-CNN. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[159] Loic Landrieu and Martin Simonovsky. 2018. Large-scale point cloud semantic segmentation with superpoint graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[160] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2014. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint arXiv:1412.6553* (2014).
[161] Vadim Lebedev and Victor Lempitsky. 2016. Fast ConvNets using group-wise brain damage. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[162] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. 2010. MNIST handwritten digit database. AT&T Labs. Retrieved from: http://yann.lecun.com/exdb/mnist.
[163] Yann LeCun, John S. Denker, Sara A. Solla, Richard E. Howard, and Lawrence D. Jackel. 1989. Optimal brain damage. In *Conference on Neural Information Processing Systems*.
[164] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. 2018. UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *International Solid-state Circuits Conference*.
[165] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*.
[166] Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2017. Simple recurrent units for highly parallelizable recurrence. In *Conference on Empirical Methods in Natural Language Processing*.
[167] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).
[168] Guohao Li, Matthias Müller, Guocheng Qian, Itzel C. Delgadillo, Abdulellah Abualshour, Ali Thabet, and Bernard Ghanem. 2021. DeepGCNs: Making GCNs go as deep as CNNs. *IEEE Trans. Pattern Anal. Mach. Intell.* (2021).
[169] Guohao Li, Guocheng Qian, Itzel C. Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2020. SGAS: Sequential greedy architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[170] Guohao Li, Mengmeng Xu, Silvio Giancola, Ali Thabet, and Bernard Ghanem. 2020. LC-NAS: Latency constrained neural architecture search for point cloud networks. *arXiv preprint arXiv:2008.10309* (2020).
[171] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient ConvNets. In *International Conference on Learning Representations*.
[172] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. 2020. GAN compression: Efficient architectures for interactive conditional GANs. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[173] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. PointCNN: Convolution on $\mathcal{X}$-transformed points. In *Conference on Neural Information Processing Systems*.
[174] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. 2020. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on Machine Learning*.

[175] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny deep learning on IoT devices. *arXiv preprint arXiv:2007.10319* (2020).
[176] Ji Lin, Chuang Gan, and Song Han. 2019. Training kinetics in 15 minutes: Large-scale distributed training on videos. *arXiv preprint arXiv:1910.00932* (2019).
[177] Ji Lin, Chuang Gan, and Song Han. 2019. TSM: Temporal shift module for efficient video understanding. In *International Conference on Computer Vision*.
[178] Ji Lin, Yongming Rao, and Jiwen Lu. 2017. Runtime neural pruning. In *Conference on Neural Information Processing Systems*.
[179] Yujun Lin, Driss Hafdi, Kuan Wang, Zhijian Liu, and Song Han. 2020. Neural-hardware architecture search. In *Workshop on ML for Systems at NeurIPS*.
[180] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2018. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*.
[181] Yujun Lin, Mengtian Yang, and Song Han. 2021. NAAS: Neural accelerator architecture search. In *Design Automation Conference*.
[182] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2016. Neural networks with few multiplications. In *International Conference on Learning Representations*.
[183] Bingbin Liu, Michelle Guo, Edward Chou, Rishab Mehra, Serena Yeung, N. Lance Downing, Francesca Salipur, Jeffrey Jopling, Brandi Campbell, Kayla Deru, William Beninati, Arnold Milstein, and Li Fei-Fei. 2018. 3D point cloud-based visual prediction of ICU mobility care activities. In *Machine Learning for Healthcare Conference*.
[184] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. 2019. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[185] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *European Conference on Computer Vision*.
[186] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*.
[187] Haoxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*.
[188] Lanlan Liu and Jia Deng. 2018. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *AAAI Conference on Artificial Intelligence*.
[189] Liu Liu, Lei Deng, Xing Hu, Maohua Zhu, Guoqi Li, Yufei Ding, and Yuan Xie. 2019. Dynamic sparse graph for efficient deep learning. In *International Conference on Learning Representations*.
[190] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by summarizing long sequences. In *International Conference on Learning Representations*.
[191] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. 2015. A recursive recurrent neural network for statistical machine translation. In *Conference of the Association for Computational Linguistics*.
[192] Yifan Liu, Ke Chen, Chris Liu, Zengchang Qin, Zhenbo Luo, and Jingdong Wang. 2019. Structured knowledge distillation for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[193] Zhijian Liu. 2020. *Hardware-efficient Deep Learning for 3D Point Cloud*. Master's. Thesis Massachusetts Institute of Technology.
[194] Zhijian Liu, Alexander Amini, Sibo Zhu, Sertac Karaman, Song Han, and Daniela Rus. 2021. Efficient and robust LiDAR-Based end-to-end navigation. In *IEEE International Conference on Robotics and Automation*.
[195] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *International Conference on Computer Vision*.
[196] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. MetaPruning: Meta learning for automatic neural network channel pruning. In *International Conference on Computer Vision*.
[197] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. 2019. Point-voxel CNN for efficient 3D deep learning. In *Conference on Neural Information Processing Systems*.
[198] Zhijian Liu, Haotian Tang, Shengyu Zhao, Kevin Shao, and Song Han. 2021. PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[199] Zhijian Liu, Zhanghao Wu, Chuang Gan, Ligeng Zhu, and Song Han. 2020. DataMix: Efficient privacy-preserving edge-cloud inference. In *European Conference on Computer Vision*.
[200] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *IEEE International System-on-Chip Conference*.
[201] Chenxu Luo and Alan L. Yuille. 2019. Grouped spatial-temporal aggregation for efficient action recognition. In *International Conference on Computer Vision*.
[202] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *European Conference on Computer Vision*.

[203] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. 2020. PCONV: The missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices. In *AAAI Conference on Artificial Intelligence*.

[204] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. 2013. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151* (2013).

[205] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani, Songtao He, Vikram Nathan, Frank Cangialosi, Shaileshh Venkatakrishnan, Wei-Hung Weng, Song Han, Tim Kraska, and Mohammad Alizadeh. 2019. Park: An open platform for learning-augmented computer systems. In *Conference on Neural Information Processing Systems*.

[206] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. 2019. Interpolated convolutional networks for 3D point cloud understanding. In *International Conference on Computer Vision*.

[207] Daniel Maturana and Sebastian Scherer. 2015. VoxNet: A 3D convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[208] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2016. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*.

[209] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Conference on Neural Information Processing Systems*.

[210] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Conference of the International Speech Communication Association*.

[211] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. 2018. A hierarchical model for device placement. In *International Conference on Learning Representations*.

[212] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*.

[213] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient transfer learning. In *International Conference on Learning Representations*.

[214] Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. 2019. K for the price of 1: Parameter-efficient multi-task and transfer learning. In *International Conference on Learning Representations*.

[215] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. In *International Conference on Computer Vision*.

[216] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.

[217] Kaoru Ota, Minh Son Dao, Vasileios Mezaris, and Francesco G. B. De Natale. 2017. Deep learning for mobile multimedia: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications* 13, 35 (2017), 1–22.

[218] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2018. OuterSPACE: An outer product based sparse matrix multiplication accelerator. In *IEEE International Symposium on High-performance Computer Architecture*.

[219] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *International Symposium on Computer Architecture*.

[220] Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. 2016. Faster CNNs with direct sparse convolutions and guided pruning. *arXiv preprint arXiv:1608.01409* (2016).

[221] Seongwook Park, Kyeongryeol Bong, Dongjoo Shin, Jinmook Lee, Sungpill Choi, and Hoi-Jun Yoo. 2015. A 1.93TOP-S/W Scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications. In *IEEE International Solid-state Circuits Conference*.

[222] Seongsik Park, Jaehee Jang, Seijoon Kim, Byunggook Na, and Sungroh Yoon. 2020. Memory-augmented neural networks on FPGA for real-time and energy-efficient question answering. *IEEE Trans. Very Large Scale Integ. Syst.* (2020).

[223] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *International Conference on Machine Learning*.

[224] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Conference on Neural Information Processing Systems*.

[225] Maurice Peemen, Arnaud A. A. Setio, Bart Mesman, and Henk Corporaal. 2013. Memory-centric accelerator design for convolutional neural networks. In *IEEE International Conference on Computer Design*.
[226] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*.
[227] A. J. Piergiovanni, Anelia Angelova, and Michael S. Ryoo. 2019. Tiny video networks. *arXiv preprint arXiv:1910.06961* (2019).
[228] A. J. Piergiovanni, Anelia Angelova, Alexander Toshev, and Michael S. Ryoo. 2019. Evolving space-time neural architectures for videos. In *International Conference on Computer Vision*.
[229] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. 2019. Deep Hough voting for 3D object detection in point clouds. In *International Conference on Computer Vision*.
[230] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. 2018. Frustum PointNets for 3D object detection from RGB-D data. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[231] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[232] Charles R. Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. 2016. Volumetric and multi-view CNNs for object classification on 3D data. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[233] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Conference on Neural Information Processing Systems*.
[234] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *IEEE International Symposium on High-performance Computer Architecture*.
[235] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. 2019. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972* (2019).
[236] Zhaofan Qiu, Ting Yao, and Tao Mei. 2017. Learning spatio-temporal representation with pseudo-3D residual networks. In *International Conference on Computer Vision*.
[237] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. (2019).
[238] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network design spaces. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[239] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing*.
[240] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).
[241] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *European Conference on Computer Vision*.
[242] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. 2020. Accelerating 3D deep learning with PyTorch3D. *arXiv preprint arXiv:2007.08501* (2020).
[243] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*.
[244] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. 2017. OctNet: Learning deep 3D representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[245] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. FitNets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).
[246] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient content-based sparse attention with routing transformers. *Trans. Assoc. Comput. Ling.* 9, 3 (2020), 53–68.
[247] Manuele Rusci, Marco Fariselli, Alessandro Capotondi, and Luca Benini. 2020. Leveraging automated mixed-low-precision quantization for tiny edge microcontrollers. arXiv preprint arXiv:2008.05124 (2020).
[248] Michael S. Ryoo, A. J. Piergiovanni, Mingxing Tan, and Anelia Angelova. 2020. AssembleNet: Searching for multi-stream neural connectivity in video architectures. In *International Conference on Learning Representations*.
[249] Tayyar Rzayev, Saber Moradi, David H. Albonesi, and Rajit Manchar. 2017. DeepRecon: Dynamically reconfigurable architecture for accelerating deep neural networks. In *International Joint Conference on Neural Networks*.
[250] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[251] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *Conference on Neural Information Processing Systems*.
[252] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf. 2009. A massively parallel coprocessor for convolutional neural networks. In *International Conference on Application-specific Systems, Architectures and Processors*.

[253] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs. In *Conference of the International Speech Communication Association*.

[254] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: An astounding baseline for recognition. *arXiv preprint arXiv:1403.6382* (2014).

[255] Sayeh Sharify, Alberto Delmas Lascorz, Kevin Siu, Patrick Judd, and Andreas Moshovos. 2018. Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks. In *Design Automation Conference*.

[256] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *International Symposium on Computer Architecture*.

[257] Sheng Shen, Alexei Baevski, Ari S. Morcos, Kurt Keutzer, Michael Auli, and Douwe Kiela. 2021. Reservoir transformer. In *Conference of the Association for Computational Linguistics*.

[258] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of BERT. In *AAAI Conference on Artificial Intelligence*.

[259] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. 2020. PV-RCNN: Point-voxel feature set abstraction for 3D object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[260] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. 2019. PointRCNN: 3D object proposal generation and detection from point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[261] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. 2020. From points to parts: 3D object detection from point cloud with part-aware and part-aggregation network. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 8 (2020), 2647–2664.

[262] Karen Simonyan and Andrew Zisserman. 2014. Two-stream convolutional networks for action recognition in videos. In *Conference on Neural Information Processing Systems*.

[263] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

[264] David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. In *International Conference on Machine Learning*.

[265] Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. In *British Machine Vision Conference*.

[266] Vinay Sriram, David Cox, Kuen Hung Tsoi, and Wayne Luk. 2010. Towards an embedded biologically-inspired machine vision processor. In *International Conference on Field-programmable Technology*.

[267] Jonathan Stroud, David Ross, Chen Sun, Jia Deng, and Rahul Sukthankar. 2020. D3D: Distilled 3D networks for video action recognition. In *IEEE/CVF Winter Conference on Applications of Computer Vision*.

[268] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Conference of the Association for Computational Linguistics*.

[269] Swathikiran Sudhakaran, Sergio Escalera, and Oswald Lanz. 2020. Gate-shift networks for video action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[270] Peng Sun, Wansen Feng, Ruobing Han, Shengen Yan, and Yonggang Wen. 2019. Optimizing network performance for distributed DNN training on GPU clusters: ImageNet/AlexNet training in 1.5 minutes. *arXiv preprint arXiv:1902.06855* (2019).

[271] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Conference on Neural Information Processing Systems*.

[272] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE* 105, 12 (2017), 2295–2329.

[273] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[274] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[275] Thierry Tambe, Coleman Hooper, Lillian Pentecost, En-Yu Yang, Marco Donato, Victor Sanh, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. 2020. EdgeBERT: Optimizing on-chip inference for multi-task NLP. *arXiv preprint arXiv:2011.14203* (2020).

[276] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, and Gu-Yeon Wei. 2020. Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In *Design Automation Conference*.

[277] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[278] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and efficient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[279] Zhanhong Tan, Jiebo Song, Xiaolong Ma, Sia-Huat Tan, Hongyang Chen, Yuanqing Miao, Yifu Wu, Shaokai Ye, Yanzhi Wang, Dehui Li, and Kaisheng Ma. 2020. PCNN: Pattern-based fine-grained regular pruning towards optimizing CNN accelerators. *arXiv preprint arXiv:2002.04997* (2020).

[280] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. 2020. Searching efficient 3D architectures with sparse point-voxel convolution. In *European Conference on Computer Vision*.

[281] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. 2018. Tangent convolutions for dense prediction in 3D. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[282] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732* (2020).

[283] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. 2017. SEGCloud: Semantic segmentation of 3D point clouds. In *International Conference on 3D Vision*.

[284] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. 2019. KPConv: Flexible and deformable convolution for point clouds. In *International Conference on Computer Vision*.

[285] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning spatiotemporal features with 3D convolutional networks. In *International Conference on Computer Vision*.

[286] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. 2019. Video classification with channel-separated convolutional networks. In *International Conference on Computer Vision*.

[287] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. 2018. A closer look at spatiotemporal convolutions for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[288] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *International Symposium on Field-programmable Gate Arrays*.

[289] Yaman Umuroglu, Lahiru Rasnayake, and Magnus Själander. 2018. BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *International Conference on Field-programmable Logic and Applications*.

[290] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Conference on Neural Information Processing Systems*.

[291] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Conference of the Association for Computational Linguistics*.

[292] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

[293] Hanrui Wang. 2020. *Efficient Algorithms and Hardware for Natural Language Processing*. Master's. Thesis Massachusetts Institute of Technology.

[294] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020. GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *Design Automation Conference*.

[295] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-aware transformers for efficient natural language processing. In *Conference of the Association for Computational Linguistics*.

[296] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. 2018. Learning to design circuits. In *Workshop on ML for Systems at NeurIPS*.

[297] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient sparse attention architecture with cascade token and head pruning. In *IEEE International Symposium on High-performance Computer Architecture*.

[298] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-aware automated quantization with mixed precision. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[299] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2020. Hardware-centric AutoML for mixed-precision quantization. *Int. J. Comput. Vis.* 128, 8 (2020), 2035–2048.

[300] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. 2016. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision*.

[301] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. In *SIGGRAPH Conference*.
[302] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. 2018. Adaptive O-CNN: A patch-based deep representation of 3D shapes. In *SIGGRAPH Asia Conference*.
[303] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Conference of the Association for Computational Linguistics*.
[304] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).
[305] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020. APQ: Joint search for network architecture, pruning and quantization policy. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[306] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. 2018. SGPN: Similarity group proposal network for 3D point cloud instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[307] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. SkipNet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision*.
[308] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic graph CNN for learning on point clouds. In *SIGGRAPH Conference*.
[309] Ziheng Wang. 2021. SparseDNN: Fast sparse deep learning inference on CPUs. *arXiv preprint arXiv:2101.07948* (2021).
[310] Zihao Wang, Chen Lin, Lu Sheng, Junjie Yan, and Jing Shao. 2020. PV-NAS: Practical neural architecture search for video recognition. *arXiv preprint arXiv:2011.00826* (2020).
[311] Zongji Wang and Feng Lu. 2019. VoxSegNet: Volumetric CNNs for semantic part segmentation of 3D shapes. *IEEE Transactions on Visualization and Computer Graphics* 26, 9 (2019), 2919–2930.
[312] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. In *Conference on Neural Information Processing Systems*.
[313] Bingzhen Wei, Mingxuan Wang, Hao Zhou, Junyang Lin, and Xu Sun. 2019. Imitation learning for non-autoregressive neural machine translation. In *Conference of the Association for Computational Linguistics*.
[314] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Conference on Neural Information Processing Systems*.
[315] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In *Conference on Neural Information Processing Systems*.
[316] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392* (2019).
[317] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[318] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.
[319] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[320] Wenxuan Wu, Zhongang Qi, and Li Fuxin. 2019. PointConv: Deep convolutional networks on 3D point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[321] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention. In *International Conference on Learning Representations*.
[322] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. 2018. BlockDrop: Dynamic inference paths in residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[323] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[324] Saining Xie, Jiatao Gu, Demi Guo, Charles R. Qi, Leonidas J. Guibas, and Or Litany. 2020. PointContrast: Unsupervised pre-training for 3D point cloud understanding. In *European Conference on Computer Vision*.
[325] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. 2018. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *European Conference on Computer Vision*.
[326] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Conference of the Association for Computational Linguistics*.

[327] Wayne Xiong, Lingfeng Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. 2018. The Microsoft 2017 conversational speech recognition system. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.
[328] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *International World Wide Web Conference*.
[329] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. 2020. Grid-GCN for fast and scalable point cloud learning. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[330] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. 2018. SpiderCNN: Deep learning on point sets with parameterized convolutional filters. In *European Conference on Computer Vision*.
[331] Jian Xue, Jinyu Li, and Yifan Gong. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Conference of the International Speech Communication Association*.
[332] Yan Yan, Yuxing Mao, and Bo Li. 2018. SECOND: Sparsely embedded convolutional detection. *Sensors* (2018).
[333] Zhongxia Yan, Hanrui Wang, Demi Guo, and Song Han. 2020. MicroNet for efficient language modeling. *J. Mach. Learn. Res.* 123, 20 (2020), 215–231.
[334] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks. In *Design Automation Conference*.
[335] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *European Conference on Computer Vision*.
[336] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. 2019. STD: Sparse-to-dense 3D object detector for point cloud. In *International Conference on Computer Vision*.
[337] Serena Yeung, N. Lance Downing, Li Fei-Fei, and Arnold Milstein. 2018. Bedside computer vision—Moving artificial intelligence from driver assistance to patient safety. *New Eng. J. Med.* 14, 378 (2018), 1271–1273.
[338] Jiahui Yu and Thomas Huang. 2019. AutoSlim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728* (2019).
[339] Jiahui Yu and Thomas S. Huang. 2019. Universally slimmable networks and improved training techniques. In *International Conference on Computer Vision*.
[340] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *International Symposium on Computer Architecture*.
[341] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable neural networks. In *International Conference on Learning Representations*.
[342] Christopher Zach, Thomas Pock, and Horst Bischof. 2007. A duality based approach for realtime TV-L1 optical flow. In *Joint Pattern Recognition Symposium*.
[343] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. GOBO: Quantizing attention-based NLP models for low latency and energy efficient inference. In *IEEE/ACM International Symposium on Microarchitecture*.
[344] Sergey Zagoruyko and Nikos Komodakis. 2017. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*.
[345] Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating neural transformer via an average attention network. In *Conference of the Association for Computational Linguistics*.
[346] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *International Symposium on Field-programmable Gate Arrays*.
[347] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *IEEE/ACM International Symposium on Microarchitecture*.
[348] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Conference on Empirical Methods in Natural Language Processing*.
[349] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Conference on Neural Information Processing Systems*.
[350] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[351] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2016. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 10 (2016), 1943–1955.
[352] Zhekai Zhang, Hanrui Wang, Song Han, and William J. Dally. 2020. SpArch: Efficient architecture for sparse matrix multiplication. In *IEEE International Symposium on High-performance Computer Architecture*.

[353] Xiangyu Zhao, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Jiliang Tang. 2020. AutoEmb: Automated embedding dimensionality search in streaming recommendations. *arXiv preprint arXiv:2002.11252* (2020).
[354] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[355] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2018. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[356] Yin Zhou and Oncel Tuzel. 2018. VoxelNet: End-to-end learning for point cloud based 3D object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[357] Chenzhuo Zhu, Song Han, Huizi Mao, and William Dally. 2017. Trained ternary quantization. In *International Conference on Learning Representations*.
[358] Ligeng Zhu, Hongzhou Lin, Yao Lu, Yujun Lin, and Song Han. 2021. Delayed gradient averaging: Tolerate the communication latency in federated learning. In *Conference on Neural Information Processing Systems*.
[359] Ligeng Zhu, Zhijian Liu, and Han Song. 2019. Deep leakage for gradient. In *Conference on Neural Information Processing Systems*.
[360] Ligeng Zhu, Yao Lu, Yujun Lin, and Song Han. 2019. Distributed training across the World. In *NeurIPS Workshop on Systems for ML*.
[361] Sijie Zhu, Taojiannan Yang, Matias Mendieta, and Chen Chen. 2020. A3D: Adaptive 3D networks for video action recognition. *arXiv preprint arXiv:2011.12384* (2020).
[362] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Flow-guided feature aggregation for video object detection. In *International Conference on Computer Vision*.
[363] Zhuotun Zhu, Chenxi Liu, Dong Yang, Alan Yuille, and Daguang Xu. 2019. V-NAS: Neural architecture search for volumetric medical image segmentation. In *International Conference on 3D Vision*.
[364] Ling Zhuo and Viktor K. Prasanna. 2005. Sparse matrix-vector multiplication on FPGAs. In *International Symposium on Field-programmable Gate Arrays*.
[365] Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.
[366] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.
[367] Dan Zou, Yong Dou, Song Guo, and Shice Ni. 2013. High performance sparse matrix-vector multiplication on FPGA. *IEICE Electron. Expr.* (2013).

Received April 2021; revised July 2021; accepted September 2021