When Deep Learning Meets the Edge: Auto-Masking Deep Neural Networks for Efficient Machine Learning on Edge Devices

Ning Lin[1,2], Hang Lu[1,2], Xing Hu[3], Jingliang Gao[1,2], Mingzhe Zhang[1,2] and Xiaowei Li[1,2]
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences[1]
University of Chinese Academy of Sciences[2]
Department of Electrical and Computer Engineering, University of California, Santa Barbara[3]
{linning19b, luhang, gaojingliang, zhangmingzhe, lxw}@ict.ac.cn, {xinghu}@ucsb.edu

***Abstract*—Deep neural network (DNN) has demonstrated promising performance in various machine learning tasks. Due to the privacy issue and the unpredictable transmission latency, inferring DNN models directly on edge devices trends the development of intelligent systems, like self-driving cars, smart Internet-of-Things (IoTs) and autonomous robotics. The on-device DNN model is obtained by expensive training via vast volumes of high-quality training data in the cloud datacenter, and then deployed into these devices, expecting it to work effectively at the edge. However, edge device always deals with low-quality images caused by compression or environmental noise pollutions. The well-trained model, though could work perfectly on the cloud, cannot adapt to these edge-specific conditions without remarkable accuracy drop. In this paper, we propose an automated strategy, called "AutoMask", to embrace effective machine learning and accelerate DNN inference on edge devices. AutoMask comprises end-to-end trainable software strategies and cost-effective hardware accelerator architecture to improve the adaptability of the device without compromising the constrained computation and storage resources. Extensive experiments, over ImageNet dataset and various state-of-the-art DNNs, show that AutoMask achieves significant inference acceleration and storage reduction while maintains comparable accuracy level on embedded Xilinx Z7020 FPGA, as well as NVIDIA Jetson TX2.**

I. Introduction

Edge devices like cellphones, unmanned aerial vehicles and autonomous robotics, are important sources for us to sense the physical world. The tremendous data collected from these devices could help us recognize the environment in a more convenient and insightful way. Traditionally, researchers are putting emphasis on cloud-based solutions to deal with the real-time collected data from these devices, seeking to rely on the powerful computation and vast storage on the cloud to accomplish deep learning tasks at a higher speed. However, due to the fluctuated network connection quality and privacy issues, it arises a growing interest in recent years in deploying deep learning directly in these devices, releasing their potentials to avoid the unpredictable data transmission latency and alleviate the burden imposed on the cloud.

Although as a critical step to migrate artificial intelligence from the cloud to the edge, the situation faced by edge devices in performing deep learning task is totally different from the situation on the cloud. Usually, training DNNs are through large scale datasets, i.e. ImageNet [2], with ideal, large amounts of high-quality images. The responsibility of the cloud is to obtain a complex and accuracy-satiable model by testing it with high-quality images as well. The second step is to deploy the well-trained model into edge devices and expect it to work properly, one-case-for-all. However, it differs from the real-world scenario in two key factors: firstly, in order to alleviate the local storage and transmission pressure, it is widely adopted that raw images collected by the front-end camera must be compressed to a certain level before handed to back-end deep learning tasks. Adopting the cloud-trained models to the compressed input images does not bring satisfying performance that can be expected on the cloud. For example, JPEG [3] is a widely used compression framework designed for trading the image quality with compromised raw sizes. The compression ratio is denoted by the "quality factor" (QF hereafter), and is tunable to balance the image quality and local storage. For human vision systems, a lower QF value that generates higher compression ratio does not impact the recognition of the images, but for the DNNs, the error rate could be as high as 62% as shown in Section II. Even for less aggressive QF (at nearly 3x compression ratio), the degradation could still reach ~8% that is even on par with the accuracy improvement by revolutionizing DNNs, i.e. from AlexNet to ResNet [4, 5, 6, 7, 8] . Secondly, the edge devices often work outside the warehouse under different illumination and noisy surroundings. For example, security cameras may encounter different extreme weathers like rain snow or fog, all of which will add environmental noises that degrade the quality of the images collected by the camera. Worse still, its impact is non-uniform and highly unpredictable, varied with time and location, which imposes a significant challenge to the edge

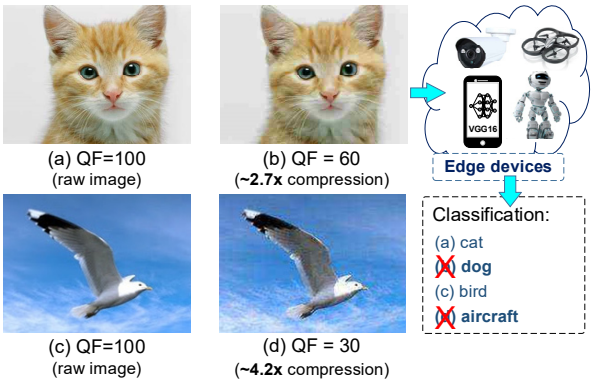

Figure 1 Typical mis-classification encountered on edge devices. Well-trained deep learning models could make wrong decisions facing these compressed images.

978-1-5386-6648-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCD46524.2019.00076

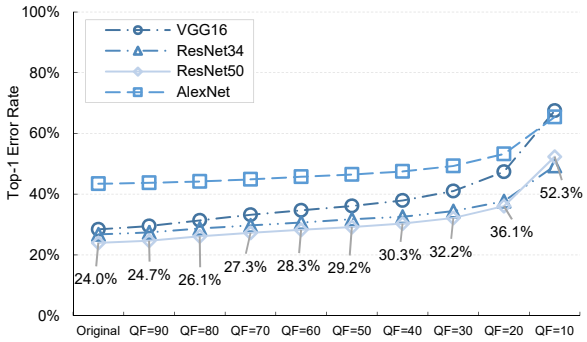

Figure 2 Top1 error rate *versus* JPEG quality factor (QF) over VGG-16 model and four state-of-the-art DNNs. The lower the quality factor (QF), the higher the compression ratio.

level intelligence – the *adaptability* problem. Adaptability means the ability for edge devices to tackle exotic interference and adapt to the environmental changes. Existing edge intelligence only focuses on the high accuracy brought by the DNN, while it lacks considerations to make the device robust enough to maintain its performance by redeeming the lost accuracy facing these *edge-specific* cases.

In essence, both front-end compressions and environmental interference will degrade the image quality, which is also the root reason that current well-trained DNN cannot work effectively, and how to release the potentials of device adaptability still has no effective solution. Therefore in this work, we propose an automated strategy, termed as *AutoMask*, to embrace effective deep learning and at the same time, accelerate the DNN inference on edge devices. AutoMask is a combination of software strategy and hardware architectures. It differs from previous approaches because it does not train and store multiple copies of the DNN for the individual use case, but employs an end-to-end trainable manner to learn an extra parameter or what we call the "mask" in this paper. Mask is just represented as 1-bit parameter associated with each weight, either bit 0 or 1, and could switch between each other based on the input image quality automatically, in order to maintain the inference accuracy under complex conditions. By supporting multiple sets of masks, the device adaptability is augmented without compromising precious resources in terms of computation and storage. Generally speaking, this paper makes the following contributions:

- *We propose an automated strategy, AutoMask, to unlock efficient machine learning at the edge.* In the software training part, we use the dataset with images pre-processed with different QFs and only train the "masks" for each QF, while the original DNN model remains intact. If we observe an accuracy degradation caused by the inferior image quality due to external factors, the masks trained for higher QF will be activated, masking the original DNN to redeem the lost accuracy. The storage only involves one copy of the original DNN and multiple copies of 1-bit masks. The tiny volume of masks introduced in our scheme can not only improve the adaptability, but also avoid burdening precious resources on edge devices.
- *We implement a specialized hardware accelerator to support AutoMask on edge devices.* In the hardware part, the accelerator is customized to handle mask based inference with minimal increased hardware cost. We implement and verify the accelerator on Xilinx Z7020 FPGA using Vivado HLS tool, and observed 1.47× inference speedup and up to 5.2× storage and 4.86× energy reduction compared with previous solutions. On GPU-based platform - NVIDIA Jetson TX2, we also observe up to 6.5× acceleration due to the high utlization of CUDA cores and high efficiency of global/shared memory accesses.

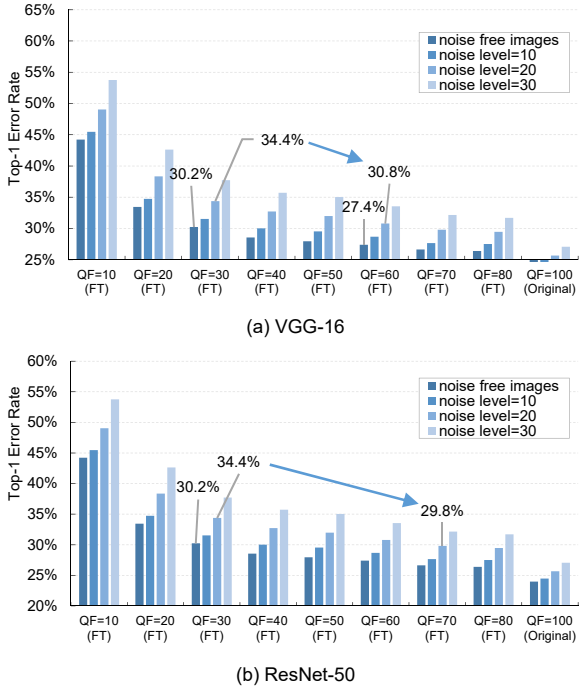

Figure 3 Top1 error rate under three noise levels. The fine-tuned (FT) DNN model for each QF responds with severe accuracy degradation under noise. We use Gaussian noise because it is the universally existing type of noise in nature [1].

The rest of this paper is organized as follows: Section II quantitatively illustrates the adaptability problem by presenting its existence over various DCNN models. Section III elaborates our methodology including software training and the accelerator design. Section IV gives the evaluations in terms of parameter scaling, the adaptability analysis, storage consumption as well as the FPGA/GPU results. Section V briefs the related work and Section VI concludes the whole paper.

II. Challenges for DNNs at the Edge

Massive images and videos are produced timely on edge devices. As the major contents to feed deep neural networks, it is widely agreed that these contents dominate the device storage, so data compression is very important to reduce the data volume to be stored and transferred. That also explains why front-end camera is usually configured taking JPEG compression with a certain QF as the major output format for image post-processing [3]. However, the compression optimized for alleviating the local storage cost would greatly degrade the performance of DNN models. As evidence, we conducted an experiment on the widely-used VGG-16 model that was already trained with high quality (QF=100) ImageNet dataset, but tested it with moderately low QF (QF=60, ~2.7 × compressed) image (Figure 1 (b)), as well as more aggressive QF=30 (~4.2× compressed) image (Figure 1 (d)). We test the model with the same original non-compressed images for comparison (Figure 1 (a) and (c)). From human vision perspective, we can conclude that (a) and (b) are both cats, and (c) and (d) are both birds, but for the DNN model, it makes wrong decisions.

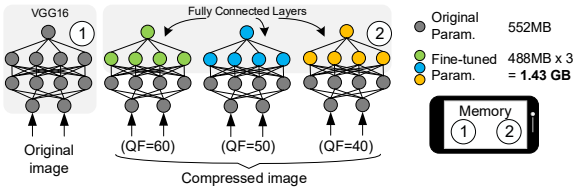

Figure 4 Concept of fine-tuning (FT) based method. The storage overhead includes the original well-trained DNN and its copies fine-tuned for each QF. We use VGG-16 as the representative.

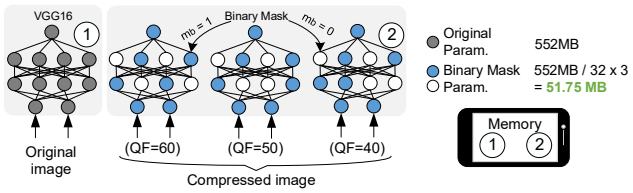

Figure 5 Concept of AutoMask. The storage overhead includes the original well-trained DNN and the binary masks lodged with each weight. The mask is a 1-bit value, either 0 or 1. We also use VGG-16 as the representative.

To make things worse, the observed wrong decision making is rather not a rare case. The error rate increase is inflicted on various state-of-the-art DNNs and scales with QF as shown in Figure 2. For VGG-16, the Top-1 error rate climbs from the original 28.41% to 47.4% at QF=20. Even for more powerful ResNet50 [6], the error rate also increases from 23.97% to 36.06%, even larger than the original accuracy of VGG-16, which means the precious accuracy improvement from VGG to ResNet has been simply overshadowed by the compressed images. Therefore, we conclude that expecting one particular cloud-trained DNN model to deal with edge-level image quality is unpractical.

As an intuitive solution to this problem, we could resort to transfer learning [9, 10] to recognize low-quality images with the same content by *fine-tuning* the original DNN with relevant low quality training dataset. As a straightforward solution, it relies on the similarity of the datasets, and the weights of lower layers, i.e. Conv1~13 in VGG-16, could remain intact while the weights of several higher layers, i.e. FC1~3 in VGG-16, are updated to suit for the new dataset. Following the same concept, we can also fine tune the original cloud-trained DNN using low-quality images until an acceptable accuracy is obtained. Although deploying the fine-tuned copy into the device could effectively deal with the accuracy degradation caused by the front-end compressions, the adaptability problem however, is still a huge barrier. Figure 3 shows the accuracy comparison under three noise polluted scenarios. We use the same test images but polluted them with three levels of Gaussian noise before feeding them to the fine-tuned DNN. There are two key observations from the figure: (1) for each QF, the error rate increases significantly with noise level varied from 10 to 30, i.e. from 35.9% to 38.1% at QF=30, from 32.2% to 34.5% at QF=60 for VGG in (a), which proves that the environmental noise is influential and should not be ignored when performing the classification task at the edge. The DNNs should not only be accurate, but most importantly, be robust to handle these cases; (2) the second observation is that different fine-tuned (FT) copies exhibit different sensitivities to the noise. As shown in (a), when the noise level is 20, the error rate at QF=30 is 38.1%; at QF=60 however, it is 34.5% which is even lower than the noise-free accuracy at QF=30 (35.9%). This observation provides a potential opportunity to improve the adaptability problem for edge devices: when the error rate increases beyond the tolerable bottom line due to environmental noises, we could redeem the accuracy by invoking the fine-tuned copy for higher QFs in the device. After the accuracy turns back to the acceptable level, we could switch back to the previous copy for the lower QF. In essence, it uses larger image size -- higher quality and larger storage, to trade higher accuracy. The same observations also appear in Figure 3 (b) for ResNet.

In order to achieve such adaptability, edge devices should support multiple FT copies and be able to dynamically switch between them according to the accuracy variation. However, storing these DNNs impose formidable burden to the hardware resources of the edge device, primarily the storage resources. If we want to store all FT copies of VGG-16 in Figure 3(a), the storage for parameters alone would reach 5.52GB (without intermediate activations accounted), despite the significant computational demand (30.7G MACs) for inference. The design goal of edge intelligent systems is supposed to be more lightweight and resource-constrained, so the appropriate solution should augment the adaptability but still comply with the low power and storage consumption. In this paper, we intend to resolve this challenge using a cost-effective, software and hardware collaborated strategy, called AutoMask, which will be elaborated in the following sections.

III. Auto-Masking Deep Neural Networks

A. Problem definition

The DNNs for image classification can be expressed as the function $\mathcal{F}(\cdot)$ mapping the input image to the output classification decision, formulated as: $y = \mathcal{F}(x, W)$ where y corresponds to the classified label of image x and W is the parameter set of the DNN model. When the input image is compressed by QF, the classification result can be expressed as:

$$y' = \mathcal{F}(x_{QF}^{compressed}, W_{QF}) \quad (1)$$

$$x_{QF}^{compressed} = JPEG(x, QF) \quad (2)$$

where $x_{QF}^{compressed}$ is the compressed image; $\mathrm{QF} = 10, 20, 30 \ldots \mathrm{N}$ denotes N quality factor candidates configured in $JPEG(\cdot)$ compression; y' is the output classification label with $x_{QF}^{compressed}$ as input and W_{QF} as parameters. In order to redeem the accuracy drop caused by the compressions, we formulated the problem as follows:

$$\text{minimize } \|y' - y\|^2 \quad (3)$$

$$\text{constrained by: } storage(W_{QF}) \leq storage(W)$$

Within our problem, the objective function is set to minimize the error between high-quality input and the compressed input. We use Euclid norm to denote this error quantitatively. The trick lies on the constraint. We need to guarantee the parameter storage for this QF is smaller than the original DNN. Thus, the parameters stored for total QFs are definitively smaller than N copies of the original DNN, which is $\mathrm{N} \times \mathrm{W}$.

B. AutoMask

1) Prerequisites

As mentioned in Section II.B, achieving the objective of Eq. (3) could resort to straightforward solutions. Intuitively, instantiating multiple copies of the original DNN, one QF apiece, and fine-

Algorithm 1 AutoMask Training Procedure. $\mathcal{C}$ is the cost function, $Binarize(m)$ specifies how to binarize mask weights, $\mathcal{F}$ is the DNN model, L is the number of layers.

Require: a minibatch of training images corresponding to a specific QF and their labels, original parameters W and the learning rate η

Ensure: trained binary mask m_b.

1. **Forward propagation**

$m_b \leftarrow Binarize(m)$

For $l = 1$ to L, compute $a_l = \mathcal{F}_l(m_b \odot w_{l-1} \cdot a_{l-1})$, with w_{l-1} and a_{l-1} already known.

2. **Backward propagation**

For $l = L$ to 2, compute δa_{l-1}, δm_b, with δa_l and m_b already known.

3. **Parameter update**

Update the real-value $m \leftarrow m - \eta \delta m_b$

tuning each copy to the target accuracy using compressed training images could simply solve this problem. However, we must consider the huge model size that is not endurable even for a single copy under the restrained storage, let alone handling multiple copies. In the FT based approach, the parameter amount of W_{QF} is always equal to W and supporting N QFs would still store N × W parameters, so it does not optimize the storage of the device. Figure 4 illustrates the concept of this method under three QF settings. If the image is of 100% high quality, the original DNN will be employed (①). When the image is compressed by QF, however, it replaces the original model with the FT model (②). Note that in this example, fine-tuning does not aim at convolutional layers but the fully connected layers (FCs), because in VGG-16 the features extracted by the former convolutional layers remain nearly the same for the compressed and non-compressed images. Even so, storing multiple copies of FCs is still a huge burden. Quantitatively, it accounts for 90% of the total parameters in VGG-16: 122M/138M. If we use the concept shown in Figure 4, it means that we need to store 122M×3 extra parameters to serve three QF levels. Transformed into actual storage u sing floating point 32 mode for each weight, it could reach 122MB×4×3=1.43GB, and might be even larger if more QFs are required. Therefore, this approach is unpractical to improve the adaptability of the device.

2) Solution Details

The key idea of AutoMask is to keep the original DNN intact, while learn an extra parameter called "mask" during training. The mask is a binary value adhering to each weight, and during inference, only the weights with mask "1" will be involved in computation. The training target this time is to obtain a set of masks that can lead to optimal accuracy for each QF. The greatest benefit is that we do not need to handle the annoying copies of huge DNNs, but only store the cost-effective masks in the device. Figure 5 shows the AutoMask concept, the original DNN model is lodged with a series of masks this time. Because each mask is only 1bit, the actual storage for each mask is 1/32 of the normal weight if we still use floating point 32 precision mode, so the size of total masks is only 51.75MB for supporting 3 QFs in the figure. The overall storage only involves the original DNN and these extra masks, which is only 552MB+51.75MB = 603.75MB in total. Compared with the FT based solution, AutoMask is obviously more efficient in storage.

AutoMask is a software and hardware collaborative approach. At the software level, it involves training the masks but the training procedure is a little bit different compared with directly training weights. As shown in Algorithm 1, we start from analyzing the forward propagation process. Masks work as independent parameters besides the normal weights, and we use matrix m to

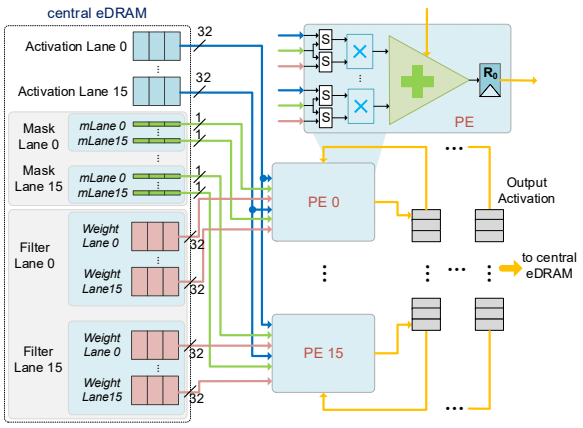

Figure 6 AutoMask Accelerator. Compared with conventional designs in the literature, the augmentation is adding the "mask" lanes, in which each element is only 1 bit.

denote the masks initialized using real values. m is then binarized by the function $Binarize(\cdot)$. The resulting matrix m_b is the binarized masks which are used for computing the output of layer l, as the forward propagation output.

In detail, we use Eq. (4) to binarize the real-value m, decided by a preset threshold τ. Then, m_b is used in the forward propagation through performing the elementwise multiplication operation $\odot$ with weights w_{l-1} and the layer input a_{l-1}, as defined in Eq. (5). These values are already obtained in advance using stochastic gradient descent (SGD) algorithm.

$$m_b = \begin{cases} 1, & if\ m \geq \tau \\ 0, & otherwise \end{cases} \tag{4}$$

$$a_l = \mathcal{F}_l(m_b \odot w_{l-1} \cdot a_{l-1}) \tag{5}$$

In the backpropagation process, however, the goal is to minimize the value of the loss function $\mathcal{C}$ by computing the gradients of a_{l-1} and m_b for layer l, denoted as δa_{l-1} and δm_b, respectively. δm_b could be easily obtained by the chained rule in backpropagation as shown in Eq. (6). The ultimate goal is to update the value of m, and we implement it through Eq. (7). η is the preset learning rate. Once m is updated, we loopback to Eq. (4) again for the iterative forward propagation.

$$\delta m_b = \frac{\partial \mathcal{C}}{\partial m_b} = \frac{\partial \mathcal{C}}{\partial a_l} \cdot \frac{\partial a_l}{\partial m_b} = \delta a_l \cdot \frac{\partial a_l}{\partial \mathcal{F}_l} \cdot \frac{\partial \mathcal{F}_l}{\partial m_b} \tag{6}$$

$$m \leftarrow m - \eta \delta m_b \tag{7}$$

It is worth noting that in Eq. (7), updating m is through the gradient of m_b (δm_b), rather than the gradient of m itself. That is because Eq. (4) is a continuous non-differentiable function. If we directly compute the derivatives of m, the result would be zero everywhere except for the threshold point τ. The incoming gradients to this threshold function would be multiplied by zero during backpropagation, so the network will learn nothing. Therefore, in order to let the gradients pass through to the input layer and make the whole process end-to-end trainable, we adopt the "straight-through estimator" method proposed in [11]. It has proved that the outgoing gradients of the threshold function could

be estimated by the incoming gradients, and the result also behaves very well [11]. We also use this trick in training AutoMask: using the gradients of thresholding masks m_b to "estimate" the gradients of the real-value masks m. By updating m using m_b, the entire system can be trained in an end-to-end differentiable manner, which makes AutoMask easy to handle from the software training perspective. The concept is very similar to training binary neural networks [12, 13, 14], but the difference is that these works directly train the binarized weights while AutoMask trains the binarized masks and keeps the original weights intact. After we obtain the binary masks m_b, we could discard the intermediate matrix m and only store the original DNN and its binary masks.

3) Accelerator Design

AutoMask requires specialized hardware accelerator when used in real-world edge devices, because the masks introduced in our scheme are also involved during inference. The major difference compared with conventional DNN accelerators [15, 16, 17, 18] is that AutoMask needs to skip over some unnecessary weights, indicated by the '0' mask. As shown in Figure 6, the '1' masks act as the enabler for the weight/activation pairs passing through to the multiple-and-accumulate (MAC) units. Mask lanes store the weight masks in the central embedded DRAM, and emit one mask element at a time to the processing element (PE). The specific 'switch module', denoted by 'S' in the figure, differentiates between mask 0 or 1 upon receiving the mask bit. The zero mask will make the switch closed and impede the incoming data proceeding to the MAC unit. At the same time, it notifies the lane to emit a new mask iteratively, until it receives a valid mask '1'.

Although the storage overhead is increased, each element in the mask lane is just one bit, either '0' or '1'. Compared with storing the whole DNN for each QF, the trivial increase in storage will not burden the device. As extra bonus, it brings with significant throughput and energy improvement. Even on a GPU-based SoC, we also observe significant performance improvement due to the efficient resource utilization. In the next section, we will thoroughly evaluate AutoMask, in terms of its accuracy, energy efficiency, storage and inference speedup over various DNNs and hardware platforms.

IV. Evaluation

Software platform. Before hardware deployment, we need to train the masks on top of the original model for each quality factor. The training procedure are implemented using PyTorch [19] on TITAN Xp GPUs, and we empirically study our method on large-scale ImageNet ILSVRC-2012 dataset [2, 4]. Various state-of-the-art DNN models, i.e. VGG, ResNet and DenseNet, are evaluated. We compare AutoMask with the fine-tuning based approach, served as our baseline and termed as "Multi-FT". For fairness, both methods use the same hyper-parameters: the optimizer is stochastic gradient descent (SGD) [20], and the epochs are set to 20 with learning rate 0.001 for the first 10 epochs and 0.0001 for the last 10 epochs. In terms of AutoMask specifics, m in Eq.(4) is initialized to 0.01 and the threshold τ is set to 5×10^{-3}. The accuracy is reported with the test dataset of ImageNet, processed under 6 QFs: 60 (~2.7× compression ratio), 50 (~3.0×), 40 (~3.4×), 30 (~4.2×), 20 (~5.1×), and 10 (~8.4×). We want to prove that by applying the trained masks to the original DNN, the accuary could be well maintained when handling low quality images. Besides, we also intentionally add Guassian noise to the test images to verify the adaptability of AutoMask. This is also a way of simulating edge devices working at outdoors. By switching to the masks for higher QF, the device is able to sustain the original performance facing noises.

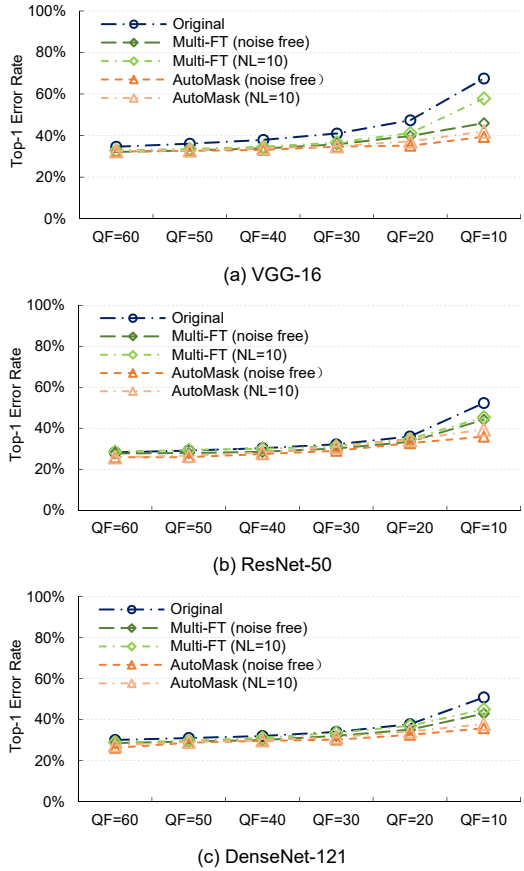

Figure 7 Top-1 error comparisons (w/ and w/o noise), tested under 6 QFs in JPEG compression.

Hardware platform. We implement AutoMask accelerator on our embedded Xilinx Z7020 FPGA, and employ Vivado HLS (v2016.2) to conduct C simulation and C/RTL hybrid simulation to extract the hardware runtime statistics like inference time, memory accesses etc. For energy consumption, we use PrimeTime tool [21] after HLS to analyze the intrinsic components of the accelerator. We instantiate 256 PEs and each PE is clocked at 125MHz. In order to evaluate its performance on GPU-based platform, we use NVIDIA Jetson TX2, an edge-level SoC, and the associate NV Profiler tool [22] to observe some key GPU performance metrics like GPU utilization, global load/store efficiency of the CPU-GPU shared memory and the warp execution efficiency. We want to show that if the user is not willing to spend time and money designing accelerator, GPU is also a feasible option for AutoMask with proved performance enhancement.

A. Accuracy and Adaptability

We firstly illustrate the platform-free results. Figure 7 shows the Top-1 error comparisons between AutoMask and the baseline. The original DNN is tested with the images of different qualities as well. We can observe that it deteriorates the accuracy with QF scaling smaller, while AutoMask can redeem the accuracy loss facing low quality images for all the evaluated DNNs. For example, at the QF=60 position of VGG model, AutoMask outperforms the original by 2.36% accuracy improvement, while at more aggressive QF=10, the improvement enlarges to 28.11%. The benefit stems from adapting compressed images by training masks for each QF, which also confirms that simply using onefold DNN

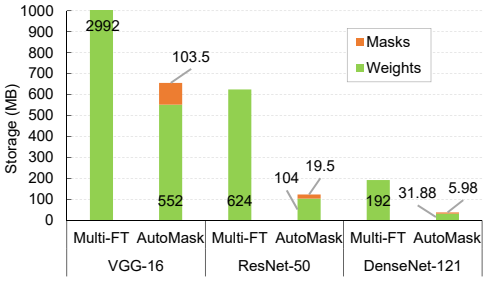

Figure 8 Storage comparisons for 6 QFs (QF=60, 50, 40, 30, 20, 10). Multi-FT does not entail masks.

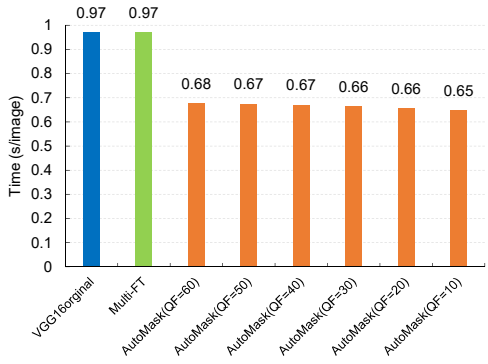

Figure 9 Comparison of actual inference time of VGG-16 model in Vivado HLS simulation on Xilinx Z7020 FPGA platform.

Table 1 Threshold parameter scaling and its impact to the final accuracy of VGG-16 model. Top-1 and Top-5 error rate vary within 1%, which means the binarization threshold setting is not critical for training masks.

τ	0.002	0.003	0.004	0.005	0.006	0.007	0.008
Top-1 (%)	66.99	67.04	67.11	67.29	67.24	66.95	67.14
Top-5 (%)	87.68	87.64	87.73	87.81	87.65	87.67	87.75

model trained only for the lossless images is inadequate to enforce effective machine learning on edge devices. AutoMask also performs better than the Multi-FT baseline: i.e. 2.33% and 1.85% lower error rate at QF=60, 8.23% and 7.17% lower error rate at QF=10, for ResNet and DenseNet respectively. Although Multi-FT targets individual QF as well, it shows that the fine-tuning based scheme is more fragile facing compressed images, while AutoMask retrains the masks from scratch and proves to be more effective.

In addition to the noise-free accuracy, we also evaluate the performance under noise pollution and verify the adaptability of AutoMask. Figure 7 demonstrates the error rate comparison with noise level set to 10. We observe that it causes trivial error rate increase at each QF: from 35.1% to 37.3% at QF=20 taking VGG as the example, but it decreases from 37.3% down to 33.9% if we switch the input from QF=20 to QF=40. Therefore, if the accuracy loss exceeds the acceptable limit, we could switch to higher QF to redeem the accuracy. In the next set of experiments, we will show that by switching the masks instead of the whole model, the alleviated storage burden makes it possible to improve the device adaptability in practical use.

B. Storage

Figure 8 shows the storage comparison between the baselines and AutoMask for 6 QFs. AutoMask exhibits 4.56× (2992MB /655.5MB), 5.05 × (624MB/123.5MB) and 5.07 × (192MB /37.86MB) less storage than Multi-FT for the three models respectively. There are two reasons for the storage improvement: first, AutoMask only stores the masks instead of multiple copies of the original DNN, so the memory consumption does not increase in proportion to the number of QFs. For the Multi-FT baseline, however, it needs to store one copy for each QF so the storage must be allocated 6× size of the original model. Secondly, the mask itself is also tiny in size. Each mask is just one bit, in conjunction with each weight, so the masks in total only occupy 15.7% portion of the total storage. Therefore, we conclude that AutoMask is more storage efficient to augment the device adaptability.

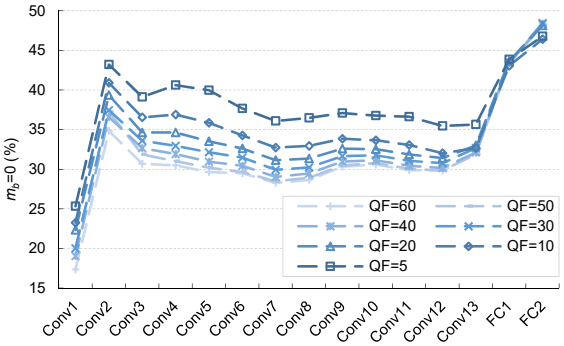

Figure 10 The layer-wise percentage of zero masks in VGG-16 model.

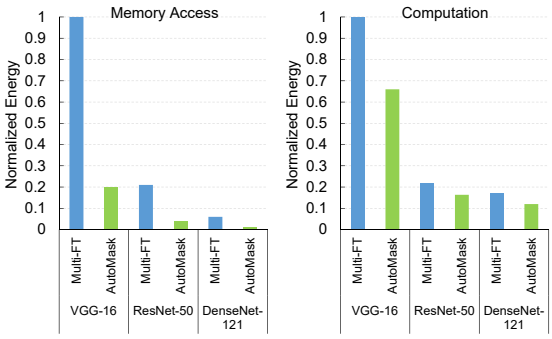

Figure 11 Comparison of the normalized energy consumption for inferring one compressed image with size 224x224.

C. Parameter Sensitivity

As the only design parameter, the threshold setting τ decides the real-value mask is binarized to 0 or 1, and further affects the training accuracy. We scale τ from 0.002 to 0.008 in training VGG-16, and the accuracy varies within 1% for both Top-1 and Top-5 as shown in Table 1, which means this parameter is not that critical in training masks. We hence uniformly use 0.005 when training the three DNNs in our evaluations.

D. FPGA Results

1) Inference Speedup.

In order to study the acceleration on edge devices, we evaluate the real-world inference time on our embedded Xilinx Z7020 FPGA. We let the accelerator infer one 224×224 image using VGG-16 model and record the inference time. As shown in Figure 9, different instances of AutoMask demonstrate slightly different speed, around 650ms~680ms, and that is because we train masks

Table 2 DNN forward propagation time (in seconds). We infer 5k images selected from ImageNet test dataset using PyTorch framework. The hardware platform is NVIDIA Jetson TX2.

DNN Models		Model scheduling (s)	Inference (s)
VGG-16	Multi-FT	61.94	525.97
	AutoMask	2.16	446.85
ResNet-50	Multi-FT	18.47	124.31
	AutoMask	0.45	122.37
DenseNet-121	Multi-FT	22.61	519.72
	AutoMask	0.88	503.96

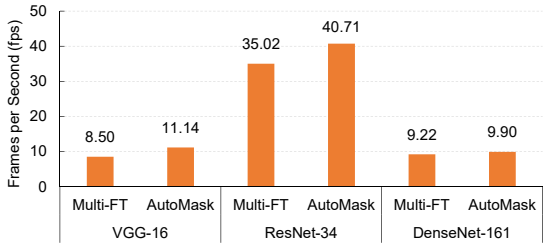

Figure 12 FPS data from NVIDIA Jetson TX2.

for each QF and the resulting masks are not identical across QFs. Compared with the inference time using original DNN (970ms), it achieves 1.47× acceleration on average due to skipping over zero masks, while Multi-FT does not exhibit any acceleration because fine-tuning operation does not reduce the computations of the DNN. We also present the layer-wise percentage of '0' masks in Figure 10 for VGG-16. Different QF configurations demonstrate similar behavior, that is, lower layers like Conv1 have relatively less 0 masks compared with deeper layers like Conv13 and the fully connected layers, indicating that the lower layers are more important that contain useful information for the accurate classification.

2) Energy Consumption

In this set of experiment, we evaluate the energy consumption under the same platform. As shown in Figure 11, we normalize the data to Multi-FT over VGG-16. AutoMask exhibits 5.0×, 5.25× and 4.62× energy reduction in memory accesses, and 1.51×, 1.34× and 1.41× reduction in computations. Because memory access dominates the overall energy consumption, the abundant energy reduction of AutoMask in this point also results in 4.93×, 5.16× and 4.45× overall energy reduction (not shown in the figure). As mentioned in Section II, less storage also leads to less energy consumption. Especially for edge devices powered by batteries, efficient energy consumption is conductive to the battery life when working outdoors and AutoMask provides both promising adaptability and endurance, thus a feasible solution for deploying machine intelligence at the edge.

E. GPU Results

1) Frames per Second (fps)

The application of masks on top of the original DNN model will breach the regularity of filters, so intuitively, we must design specialized accelerators as the only way to accelerate DNN inference. However, for the users who are reluctant to afford the expense, an alternative plan is supposed to apply AutoMask on off-the-shelf GPGPU-based platform. Amazingly, we also observe substantial acceleration in supporting the adaptability on GPU-based SoCs. In this experiment, we use NVIDIA Jetson TX2 as the representative, and mimic a real-world scenario by dynamically

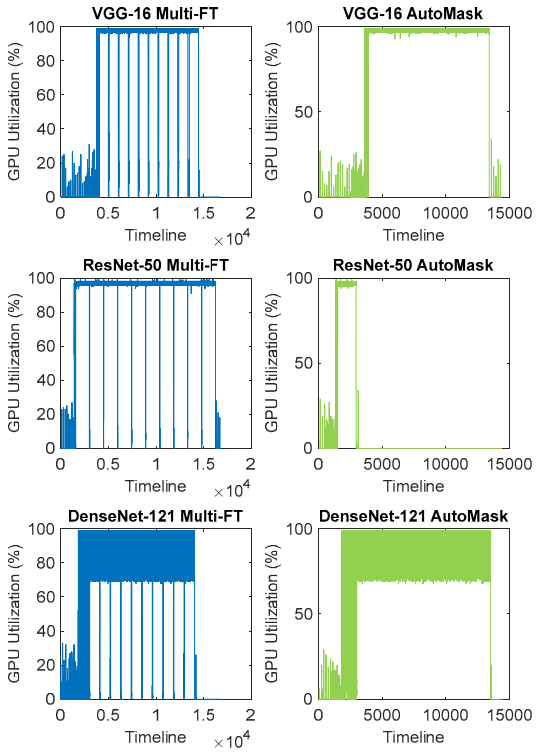

Figure 13 GPU utilization obtained via NV profiler. We only care about the region of interest, in which Multi-FT exhibits frequent vibrations from the top utilization to the bottom, while AutoMask does not. The utilization drop is caused by the model switching to redeem the degraded accuracy.

switching the models iteratively in the device to deal with the accuracy drop due to the noise-polluted input images. As shown in Table 2, we select 5000 images from ImageNet test dataset, and after inferring every 100 images we switch to another QF to seemingly redeem the classification accuracy. After continuously inferring all the test images, we record and compare the overall inference time. We can see that the model switching time of Multi-FT is very expensive: 28.7× more than AutoMask for VGG-16, and even 44.0× more for ResNet-50. The reason is that model switching involves interaction with the device memory, storage and the file system. Multi-FT switches the whole model which is very huge in volume, and moving the vast number of parameters into the device memory is very time consuming. By sharp contrast, AutoMask only switches the cost-effective masks, and the switching time only occupies 0.04%, 0.03% and 0.01% over total time for the three DNNs respectively. The results in Figure 12 also confirm that AutoMask has higher frames per second (fps). Even used on GPU-based platform, it is also more feasible in real-time machine learning applications.

2) Efficiency

GPU Utilization. We use NV profiler to study the runtime GPU utilization, as shown in Figure 13. Under the same configuration as the previous experiment, we find that CUDA cores are frequently interrupted by model switching in Multi-FT. The utilization fluctuates between ~95% and ~1%. AutoMask does not exhibit obvious fluctuation, and the utilization stables at ~95%. This again confirms that if we want to improve the adaptability of the device by supporting multiple QFs, instantiating equivalent

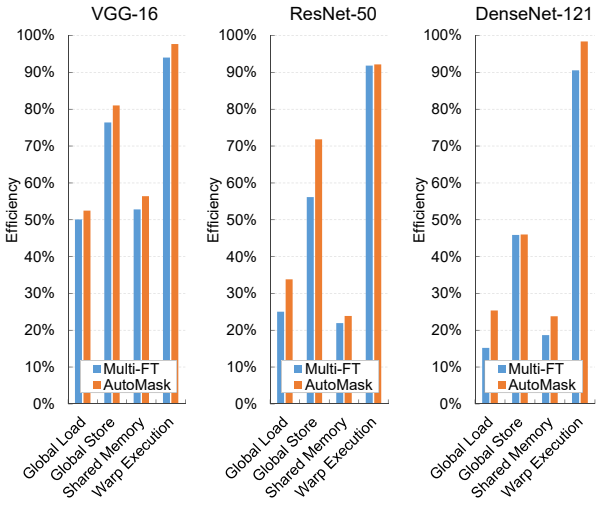

Figure 14 GPU efficiency comparison. We select 4 key metrics reflecting the efficiency of global DRAM load/store, shared memory and warp execution.

number of FT copies is suboptimal w.r.t. the device storage and even computation.

Memory Efficiency. In terms of the memory efficiency, we use four metrics: global load efficiency, global store efficiency, shared memory efficiency and warp execution efficiency. The former three metrics represent the ratio of *requested* memory operation throughput to the *required* memory operation throughput, expressed as percentage, while the last metric represents the ratio of the average active threads per warp to the maximum number of threads per warp supported on a CUDA core [23]. For the three evaluated DNNs shown in Figure 14, AutoMask outperforms Multi-FT in all four metrics; especially for the global load, AutoMask does not need to load voluminous parameters because the original parameters are shared by different set of masks, so it does not impose significant global load requests from each warp that may outweigh the memory access bandwidth.

V. Related Work

A plethora of work are dedicated to the high-accuracy image classification, and most of them try to propose novel DNN models with deeper and more complex layers [5, 6, 24]. Although these models perform well in boosting the ever increasing accuracy, none of them focuses on the adaptability behavior when these DNNs are deployed on edge devices. AutoMask is designed to adapt to different image conditions polluted by environmental noise or intentional compression like JPEG, in order to redeem the lost accuracy and at the same time alleviate the storage burden that is critical to the battery powered edge devices. It is applicable across a wide range of DNN models, from the huge models commonly used in object detection and semantic segmentation, i.e. VGG-16, VGG-19 [5], to small compact models used in lightweight image classification, i.e. MobileNet [25], ResNet-18 [6] and DenseNet-121 [8].

Some other quantization schemes like training binary [14, 26], ternary [27] and even power of two weights [28] are proposed to reduce the parameter size. Although these schemes could alleviate the storage burden, the price is the severe accuracy loss. Even if we use the most state-of-the-art binary quantization [29], the accuracy still drops by 5.8% on top of the non-quantized ResNet-18 model, and these schemes are not designed for descent adaptability as well. There are also no data reported when these quantized models are tested with low quality images. However, it has been proved that our AutoMask could maintain the lossless accuracy using comparable storage space, and most importantly it resolves the adaptability problem when the device is facing low quality images.

Some recently proposed pruning schemes [30, 31, 32, 33] could blank out trivial weights but still keeps the accuracy nearly intact. Intuitively, we could use pruning to reduce the model size, which has potentials in dealing with multiple quality levels of the input images. However, pruning also relies on iterative FTs until the accuracy is satiable, which means different quality levels require their respective pruned model to suit for each case. Therefore, the pruned models in total also consume significant storage as the FT-based approach. To make things worse, the extra indices also need to be stored and fetched together with the weights for inference, so the actual storage demand might be even larger. Comparatively, AutoMask could achieve both lossless accuracy and minimally reduced storage at the same time, by only applying different set of masks without pruning or quantizing the original model.

VI. Conclusion

In this paper, we propose a novel method -- *AutoMask* to make machine learning efficiently work at the edge. Targeting the accuracy drop due to the compressed images that are frequently encountered on edge devices, we mask the original DNN model to automatically identify computable parameters and skip over unnecessary parameters under different use cases, without compromising the key design constraints like accuracy, storage, inference speed and the energy consumption. It provides a unique opportunity to improve the adaptability of the device, by switching to a set of masks trained for higher level QFs in order to redeem the accuracy drop caused by the environmental noises. We also design a specialized hardware accelerator on embedded FPGA platform to support AutoMask in DNN inference. As an extra bonus, if the user is not willing to design specific accelerator for existing edge devices, abundant acceleration is also observed on GPU-based platform. We hope our proposed scheme could bring up new considerations in deploying effective machine learning in future intelligent systems like cyber-physical systems, internet-of-things, as well as autonomous robotics.

Acknowledgement

This work is supported in part by the national Natural Science Foundation of China (NSFC) under grant No. (61432017, 61602442, 61876173), and in part by the National Key Research and Development Project under grant No. 2018AAA0102700. Corresponding authors are Hang Lu and Xiaowei Li.

References

[1] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing,* vol. 26, no. 7, pp. 3142-3155, 2017.
[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[3] G. K. Wallace, "The JPEG still picture compression standard," *IEEE transactions on consumer electronics,* vol. 38, no. 1, pp. xviii-xxxiv, 1992.
[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[7] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.
[8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.

[9] *CS231n Convolutional Neural Networks for Visual Recognition*. Available: http://cs231n.github.io/transfer-learning/
[10] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *NIPS*, 2014, pp. 3320-3328.
[11] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432,* 2013.
[12] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to+ 1 or− 1. arXiv: 1602.02830, 2016," ed, 2017.
[13] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830,* 2016.
[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.
[15] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014, pp. 609-622.
[16] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ISCA*, 2016, vol. 44, no. 3, pp. 367-379.
[17] H. Lu, X. Wei, N. Lin, G. Yan, and X. Li, "Tetris: re-architecting convolutional neural network computation for machine learning accelerators," in *ICCAD*, 2018.
[18] A. Parashar *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, 2017.
[19] *PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration*. Available: http://pytorch.org/
[20] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*: Springer, 2010, pp. 177-186.
[21] *PrimeTime Static Timing Analysis*. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html
[22] *NVPROF: CUDA Toolkit Documentation.* Available: http://docs.nvidia.com/cuda/profiler-users-guide/index.html
[23] *Devices with compute capability 6.x implement the metrics*. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference-6x
[24] S. Xie, R. B. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," *arXiv: Computer Science,* 2017.
[25] A. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv: Computer Vision and Pattern Recognition,* 2017.
[26] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.
[27] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711,* 2016.
[28] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044,* 2017.
[29] S. Ye *et al.*, "Progressive DNN Compression: A Key to Achieve Ultra-High Weight Pruning and Quantization Rates using ADMM," *arXiv: Neural and Evolutionary Computing,* 2019.
[30] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.
[31] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710,* 2016.
[32] N. Lin, H. Lu, X. Wei, and X. Li, "HeadStart: Enforcing Optimal Inceptions in Pruning Deep Neural Networks for Efficient Inference on GPGPUs," in *DAC*, 2019.
[33] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017.