EtinyNet: Extremely Tiny Network for TinyML

Kunran Xu[1,2], Yishi Li[1,2], Huawei Zhang[1,2], Rui Lai[1,2*], Lin Gu[3,4*]
[1]School of Microelectronics, Xidian University, Xi'an 710071, China
[2]Chongqing Innovation Research Institute of Integrated Cirtuits, Xidian University, Chongqing 400031, China.
[3]RIKEN AIP, Tokyo103-0027, Japan
[4]The University of Tokyo, Japan
aazzttcc@gmail.com, yshlee1994@outlook.com, myyzhww@gmail.com
rlai@mail.xidian.edu.cn, lin.gu@riken.jp

Abstract

There are many AI applications in high-income countries because their implementation depends on expensive GPU cards (∼2000$) and reliable power supply (∼200W). To deploy AI in resource-poor settings on cheaper (∼20$) and low-power devices (<1W) , key modifications are required to adapt neural networks for Tiny machine learning (TinyML). In this paper, for putting CNNs into storage limited devices, we developed efficient tiny models with only hundreds of KB parameters. Toward this end, we firstly design a parameter-efficient tiny architecture by introducing dense linear depthwise block. Then, a novel adaptive scale quantization (ASQ) method is proposed for further quantizing tiny models in aggressive low-bit while retaining the accuracy. With the optimized architecture and 4-bit ASQ, we present a family of ultra-lightweight networks, named EtinyNet, that achieves 57.0% ImageNet top-1 accuracy with an extremely tiny model size of 340KB. When deployed on an off-the-shelf commercial microcontroller for object detection tasks, EtinyNet achieves state-of-the-art 56.4% mAP on Pascal VOC. Furthermore, the experimental results on Xilinx compact FPGA indicate that EtinyNet achieves prominent low power of 620mW, about 5.6 × lower than existing FPGA designs. The code and demo are in https://github.com/aztc/EtinyNet

Introduction

Tiny machine learning (TinyML), executing AI workloads locally on low-cost and low-energy hardwares, has grown rapidly in recent years. As shown in Fig 1, TinyML generally processes data near sensors with only hundreds of milliwatts of power while running Convolutional Neural Networks (CNNs) on IoT devices, *e.g.*, microcontroller unit (MCU) and field programable gate array (FPGA). It's a fresh and attractive area different from well developed MobileML and CloudML (Banbury et al. 2020). TinyML is believed to make a broad of new applications possible, including smart manufacturing, consumer electronics, precision agriculture, wildlife conservation and many other domains.

However, TinyML presents severe challenges, as CNNs usually require too many parameters to be deployed on IoT devices with limited storage. For example, the state-of-the-art STM32F746 (ARM Cortex-M7 CPU) MCU has only

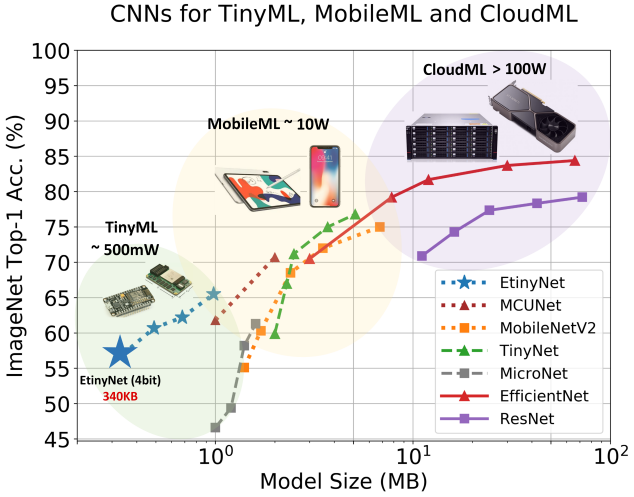

Figure 1: The model size and ImageNet top-1 accuracy of variety CNN models for TinyML, MobileML and CloudML. Different from MobileML and CloudML that there are abundant hardware resources for running large models, TinyML denotes the scenarios in which storage and power are strictly limitd, generally of < 1MB and hundreds of milliwatts.

320KB on-chip SRAM and 1MB Flash, which shows a big gap between the desired and available hardware capacity: ResNet50 (He et al. 2016) exceeds the storage limit by 100 × while the 8-bit quantized MobileNetV2 (Sandler et al. 2018) still exceeds the storage by 2.5 ×. Moreover, transmission massive model parameters between different hierarchies of memory (*e.g.* from DRAM/Flash to SRAM or from SRAM to register) also results in considerable energy consumption. It is reported that FPGAs need 3.5W - 9W of power to run CNNs, nearly 80% of which is caused by transmission of model parameters (Yu et al. 2020; Guo et al. 2018; Xiao et al.; Lian et al. 2019), far exceeding the requirements of TinyML. The large model size of existing CNNs brings much difficulties for TinyML.

Despite the recent efficient lightweight CNN architectures (Sandler et al. 2018; Li et al. 2021; Han et al. 2020) for MobileML as depicted in Fig 1, it is still hard for them to address the challenges presented by TinyML. Most of

*Corresponding author.

the above-mentioned methods are designed to accelerate the inference on embedded GPUs or smartphones. Therefore, they generally aim for reducing the Multiply-Adds (MAdds) rather than parameters of models. For example, the state-of-the-art MicroNet-M1 (Li et al. 2021) reaches ultra low MAdds of 6M but has 1.8M parameters, which still struggles to fit the requirement of most IoT edge devices. Recently, a series of research succeed in deploying CNNs on MCUs (Banbury et al. 2020; Lin et al. 2020; Sudharsan, Breslin, and Ali 2021; Zim 2021; Chappa and El-Sharkawy 2020) by greatly shrinking the model size, however, at the cost of accuracy.

In this paper, we will focus on building the CNN models with minimal parameters for high efficiency. Firstly, we present novel dense linear depthwise block to improve the existing CNN architectures, which reduces the information loss by removing the non-linear function behind depthwise convolution. It also increases the equivalent width of model via dense connections. Building upon this block, we arrive at the tiny model EtinyNet, achieving state-of-the-art 65.5% ImageNet top-1 accuracy with only 0.98M parameters. To further reduce the model size, we propose an innovative adaptive scale quantization (ASQ) method for quantizing tiny models in aggressive low-bit. Since there are much less redundancy parameters in EtinyNet, directly applying existing weight quantization DoReFa (Zhou et al. 2016) scheme would result in significant accuracy drop. By investigating the accuracy degradation mechanism, we introduce an adaptive re-scaling strategy to balance the quantization error and information entropy for retaining model accuracy. With 4-bit ASQ, EtinyNet is further compressed to a half and reaches an unprecedented tiny size of 340KB. More than that, this extremely tiny model achieves up to 57.0% ImageNet top-1 accuracy, which exhibits its excellent parameter efficiency.

We have also verified the effectiveness of the proposed method in practical TinyML applications by deploying EtinyNet on MCUs and FPGAs. Object classification results show that EtinyNet achieves a record ImageNet top-1 accuracy of 65.8% on STM32F746 MCU. On object detection tasks, EtinyNet obtains 56.4% mAP on Pascal VOC (Everingham et al. 2009) dataset, outperforming the state-of-the-art MCUNet (Lin et al. 2020) by remarkable 5% . When deploying EtinyNet on FPGA, less model parameters result in significant reduction in transmission latency and energy consumption. As a result, we succeed in reducing the power of FPGA to 620mW, nearly 5.6 $\times$ lower than existing designs, and realizing a high throughput of 103FPS.

In summary, we make the following contributions:

1) An extremely tiny CNN architecture named EtinyNet which is specially designed for the high parameter efficiency. Compared with existing lightweight CNN models, EtinyNet obtains higher performance with far less parameters.
2) A novel adaptive scale quantization (ASQ) method for quantizing tiny models. ASQ method significantly alleviates the side effect of existing DoReFa quantization scheme and, as a result, greatly improves the accuracy of tiny quantized models.
3) Based on the proposed EtinyNet and ASQ quantization method, we promote CNNs to achieve state-of-the-art processing efficiency on kinds of IoT edge devices, greatly advancing the TinyML community.
4) Our proposed TinyML solution, deployed on cheaper ($\sim$20\$) and lowpower devices ($<$1W), is essential step to let AI contribute to human well-being in resource-poor settings.

Related Works

Here we revisit the related works including efficient neural network design, low-bit quantization and CNNs for TinyML.

Efficient Neural Network Design. Efficient neural network design aims to construct the CNN architecture with lower computation, latency and energy consumption. Toward this end, the residual connection (He et al. 2016), depthwise-separable convolution (Howard et al. 2017), densely-connected block (Huang, Liu, and Weinberger 2017), and many other efficient building blocks have been introduced into CNNs design space, which greatly compressed CNN models. To further reduce the model redundancy, a series of pruning methods (Han, Mao, and Dally 2016; Li et al. 2017; Tang et al. 2020; Luo, Wu, and Lin 2017) have been successively proposed to remove the connectivity that are insensitive to the model performance. Recently, neural architecture search (NAS) (Zoph and Le 2017; Zoph et al. 2018; Liu, Simonyan, and Yang 2019; Tan et al. 2019; Howard et al. 2019; Han et al. 2020) becomes an much attractive topic, which ferrets out a mount of well-known architectures, *e.g.* EfficientNet (Tan and Le 2019), MobileNetV3 (Howard et al. 2019) and TinyNet (Han et al. 2020), *etc*. These methods succeed in reducing the computation complexity of CNNs to $<$ 30M MAdds and enable the CNNs inference on mobile phone less than 10ms. Although achieving impressive results, the above-mentioned methods are generally devoted to reduce MAdds instead of resolving the storage challenge in IoT edge devices.

Low-bit Quantization. CNNs are generally trained using single-precision floating point values. Low-bit quantization represents parameters or activations using low-bit values in the inference phase to reduce model size and computation complexity. Among kinds of quantization methods, most of them focus on minimizing the difference in values (Rastegari et al. 2016) or distributions (Choi et al. 2018; Han, Mao, and Dally 2016) between quantized weights/activations and full-precision ones. Others (Zhang et al. 2018; Wang et al. 2019) seek for developing learning-based quantizer that is trained from data, for mixed-precision quantization. For better estimation of the gradients of rounding function, (Bai, Wang, and Liberty 2019) proposed a regularizer based quantization method while (Gong et al. 2019) suggested a differentiable soft quantization. These methods reduce the gradient error caused by the straight through estimation (STE) (Bengio, Léonard, and Courville 2013) and thus improving the training stability. Recently, (Jin, Yang, and Liao 2019) improved quantization performance from the perspective of efficient training and achieved superior

results. Nevertheless, above-mentioned methods mainly aim at the quantization on standard CNN architectures of normal sizes but pay little attention to the quantization on tiny models, which is a more challenging task due to less redundancy parameters involved in models.

CNN for TinyML. TinyML is a fast-growing area. Recently, the continuously emerging studies on TinyML allow deploying CNNs on MCUs. Many works are devoted to build memory-efficient inference engines for rapidly deploying CNNs on various MCUs such as TensorFlow Lite Micro (David et al. 2020), CMSIS-NN (Lai, Suda, and Chandra 2018), CMix-NN (Capotondi et al. 2020), and MicroTVM (Chen et al. 2018). These frameworks usually interpret the network graph at runtime, which will consume a lot of SRAM and Flash. To overcome this, MicroNets (Banbury et al. 2020) employs differentiable NAS to search for models with low memory usage and low op count as well as reduce the model latency of running on MCUs. MCUNet (Lin et al. 2020) is further proposed to jointly design a more efficient CNN architecture as well as its corresponding lightweight accelerating engine, which realizes ImageNet-scale inference on STM32F746 MCU with 63.5% Top-1 accuracy. In this paper, we not only seek for the better processing efficiency on MCUs but also investigate the feasibility of running CNNs on FPGA at milliwatts of power.

Methods

In this section, we firstly introduce the proposed parameter-efficient CNN model named EtinyNet and then elaborate on the adaptive scale quantization.

EtinyNet

Depthwise-Separable Convolution. To reduce the computations and parameters, MobileNet proposed the depthwise-separable convolution made up of two operations: depthwise convolution and pointwise convolution (1×1 convolution). The depthwise convolution applies a single filter on each input channel while the pointwise convolution is then used to create a linear combination of the depthwise layer outputs. Let $\mathbf{I} \in \mathcal{R}^{C\times H\times W}$ and $\mathbf{O} \in \mathcal{R}^{D\times H\times W}$ respectively represent the input feature maps and output feature maps, depthwise-separable convolution can be computed as:

$$\mathbf{O} = \sigma(\phi_p(\sigma(\phi_d(\mathbf{I})))), \tag{1}$$

where ϕ_d, ϕ_p represent the depthwise convolution and pointwise convolution while σ denotes the non-linearity activation function, *e.g.* ReLU. By decoupling spatial-correlation and channel-correlation, depthwise-separable convolution reduces the computations and parameters by about $9\times$. It is now the key building block for lightweight CNN architectures.

Linear Depthwise Block. It has been demonstrated in (Sandler et al. 2018) that ReLU in bottleneck block would prevent the flow of information and thus impair the capacity as well as expressiveness of model. We further observed that the ReLU behind depthwise convolution also harms the model accuracy. In view of this, we remove the ReLU behind depthwise convolution and rewrite the linear depthwise-separable convolution as:

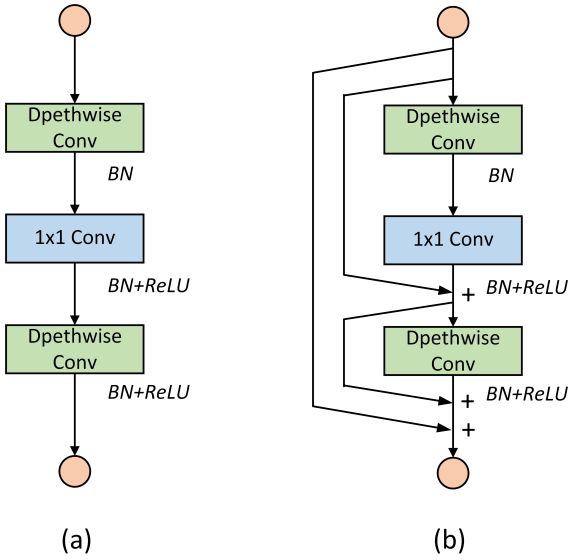

Figure 2: The proposed building blocks that make up the EtinyNet. (a) is the linear depthwise block (LB) and (b) is the dense linear depthwise block (DLB).

$$\mathbf{O} = \sigma(\phi_p(\phi_d(\mathbf{I}))). \tag{2}$$

Note that since ϕ_d and ϕ_p are both linear, these two functions now can be merged into one larger linear transformation. In other words, there exists a standard convolution ϕ_s that can be rigorously decomposed to ϕ_d and ϕ_p, which is a specific case of sparse coding (Lee et al. 2006). We will empirically validate that the removal of ReLU behind depthwise convolution helps to achieve higher parameter efficiency. In existing lightweight models, depthwise convolution layers generally possess about only 5% of the total parameters but contribute greatly to model accuracy, which indicates depthwise convolution is with high parameter efficiency. Taking advantage of this, we introduce more depthwise convolutions into the architecture by using:

$$\mathbf{O} = \sigma(\phi_{d2}(\sigma(\phi_p(\phi_{d1}(\mathbf{I}))))). \tag{3}$$

where ϕ_{d2} is another depthwise-convolution. The resulting structure is denoted as the linear depthwise block (LB) shown in Fig 2(a). The proposed LB of *dconv-pconv-dconv* is different from the commonly used bottleneck block of *pconv-dconv-pconv* in other lightweight models, for depthwise convolution accounting for larger proportion of model parameters. For the width of C, the ratio of depthwsie parameter to pointwise parameters in LB is $(C \times 9 \times 2)/(C \times C) = 18/C$ while that of bottleneck block is $(C \times 9)/(C \times C \times 2) = 4.5/C$. We argue that the higher ratio of depthwise parameters is beneficial to the model parameter efficiency, which is essential for tiny models.

Dense Connection. Restricted by the total number of parameters, the width of network can not be too large. How-

layer	size	EtinyNet-1.0	EtinyNet-0.75
Conv	112^2	3×3, 32, stride 2	3×3, 24, stride 2
Pool	56^2	2×2, max, stride 2	
LB*	56^2	[32,32,32] $\times$ 4	[24,24,24] $\times$ 4
LB	28^2	[32,128,128] $\times$ 1 [128,128,128] $\times$ 3	[24,96,96] $\times$ 1 [96,96,96] $\times$ 3
DLB	14^2	[128,192,192] $\times$ 1 [192,192,192] $\times$ 2	[96,168,168] $\times$ 1 [168,168,168] $\times$ 2
DLB	7^2	[192,256,256] $\times$ 1 [256,256,256] $\times$ 1 [256,512,512] $\times$ 1	[168,192,192] $\times$ 1 [192,192,192] $\times$ 1 [192,384,384] $\times$ 1
	1^2	7×7, globalavg, FC-1000	
MAdds		117M	75M
Params.		976K	680K

Table 1: Configurations of the proposed EtinyNet-1.0 and EtinyNet-0.75. * denotes the stage of maximum memory consumption for storing feature maps. The number in [] indicates the channels of each convolution in LB or DLB.

ever, width of CNNs is important for achieving higher accuracy (Zagoruyko and Komodakis 2016). As suggested by (Wu, Shen, and Hengel 2019), the structure with shortcut connection could be regarded as a wider network consisting of sub-networks. Therefore, we introduce the dense connection into LB for increasing its equivalent width. We refer the resulting block to dense linear depthwise block (DLB), which is depicted in Fig 2(b). Note that we take the ϕ_{d1} and ϕ_p as a whole due to the removal of ReLU, and add the shortcut connection at the ends of these two layers.

EtinyNet Architecture. By stacking the LB and DLB, we build two architectures of EtinyNet-1.0 and EtinyNet-0.75 as indicated in Table 1. Since dense connection requires storing more feature maps (the first input feature maps must be preserved until addition) than plain layers during inference, we only utilize DLB at the high level stages, in which the feature maps have much smaller size, for saving the runtime memory consumption. $*$ in Table 1 indicates the stage of maximum memory consumption for storing feature maps, which are only 245KB and 220.5KB (quantized in 8-bit) for EtinyNet-1.0 and EtinyNet-0.75. Consider that most IoT devices only have $<$ 512KB on-chip SRAM, the low runtime memory consumption makes EtinyNet easier to be deployed. It's encouraging that EtinyNet-1.0 has only 987K parameters while EtinyNet-0.75 acquires an ultra tiny size of 680K parameters. In addition, we don't use hard-swish activation (Howard et al. 2019) and squeeze-and-excitation block (Hu et al. 2020) since they are not efficiently supported by storage restricted IoT devices.

Adaptive Scale Quantization

The most common weight quantization method is the DoReFa scheme, which involves two steps of clamping and quantization. Clamping transforms the weights to values between -1 and 1, meanwhile, quantization rounds the weights to integers. For a weight matrix $\mathbf{W}$, the clamping step can

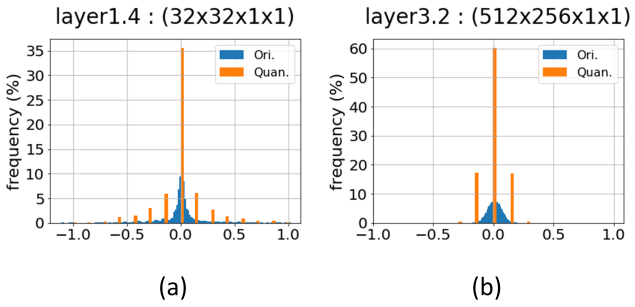

Figure 3: The frequency histogram of model parameters in EtinyNet-1.0. (a) denotes the distribution of pointwise convolution weights in layer1.4, whose shape is $32 \times 32 \times 1 \times 1$. (b) denotes the distribution of pointwise convolution weights with shape of $512 \times 256 \times 1 \times 1$ in higher level layer. Ori. indicates the original distribution while Quan. is the distribution after quantization. It is clear that DoReFa scheme introduces significant quantization error.

be represented as:

$$\hat{\mathbf{W}} = \frac{1}{2}\left(\frac{tanh(\mathbf{W})}{max(|tanh(\mathbf{W})|)} + 1\right), \tag{4}$$

where $\hat{\mathbf{W}}$ denotes the clamped weight. This transformation generally contracts the scale of large weights, and enlarges the difference of small scale elements. With weights clamped to [-1, 1], the linear quantization is further imposed on $\hat{\mathbf{W}}$ with the following function:

$$\mathbf{Q} = \frac{2}{a}\lfloor a\hat{\mathbf{W}} \rceil - 1. \tag{5}$$

where $\mathbf{Q}$ denotes the quantized weights, $\lfloor \cdot \rceil$ indicates rounding to the nearest integer and a equals $2^b - 1$, where b stands for the number of quantization bits.

DoReFa scheme works well when quantization bits is relatively higher (8-bit or 16-bit), but it would lead to significant quantization error when extremely low bit-width is applied, such as 4-bit. To show the quantization error intuitively, we plot the frequency histogram of weights $\mathbf{W}$ as well as the corresponding 4-bit quantized weights $\mathbf{Q}$. Fig 4(a) illustrates the distribution of pointwise convolution weights in layer1.4 of EtinyNet-1.0, which contains 1024 ($32 \times 32 \times 1 \times 1$) elements. It is clear that there is a big gap between original distribution and quantized distribution. A large proportion of values distributed around mean are collapsed to zero after quantization, resulting in significant information loss. As suggested by previous works (Jin, Yang, and Liao 2019, 2020), this would lead to considerable accuracy degeneration of models. When it comes to higher level layer (pointwise convolution in layer3.2), the parameters have a more centralized distribution, mainly concentrating in [-0.25, 0.25] with the standard deviation of 0.054, which is harder to quantize. Nearly 60% of the parameters become zero due to the limited quantization precision of only $\frac{1}{15}$, causing really low information entropy and parameter efficiency. More than that, DoReFa cannot take full advantage

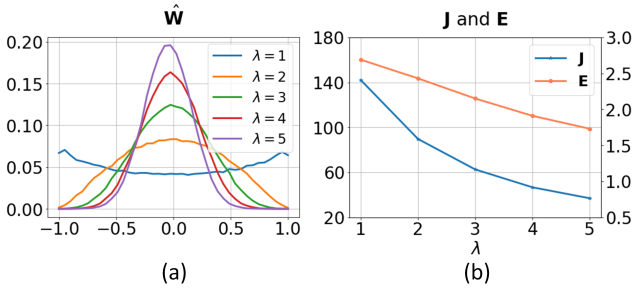

Figure 4: The distribution of re-scaled weights of pointwise convolution in layer3.2 (a) as well as the quantization error and information entropy using ASQ. As λ increases from 1 to 5, weights becomes more centralized, quantization error and information entropy decrease simultaneously.

of the dynamic range of 4-bit representation. There are only five valid quantized values out of the total quantization levels of 16 as shown in Fig 3, which further reduces the parameter efficiency. We empirically find that the quantization error of 4-bit DoReFa results in severe accuracy degeneration, especially for tiny models. Since there are much less redundant parameters of the tiny model, DoReFa scheme would significantly cut down valid parameters, and greatly reduce the model capacity.

To alleviate this side effect of DoReFa, we propose the novel Adaptive Scale Quantization (ASQ) by replacing the clamping step and quantization step with following:

$$\begin{aligned} \tilde{\mathbf{W}} &= \frac{\mathbf{W}}{\lambda\sqrt{\mathcal{VAR}(\mathbf{W})+\epsilon}}, \\ \hat{\mathbf{W}} &= \frac{tanh(\tilde{\mathbf{W}})}{max(|tanh(\tilde{\mathbf{W}})|)}, \\ \mathbf{Q} &= \frac{1}{\hat{a}}\lfloor \hat{a}\hat{\mathbf{W}} \rceil. \end{aligned} \tag{6}$$

where $\mathcal{VAR}(\cdot)$ denotes the variance function, $\epsilon = 1e-5$ prevents dividing by zero and the parameter $\lambda \in \mathbb{R}^+$ re-scale the weights to $\tilde{\mathbf{W}}$. The symmetrical quantization is utilized so that $\hat{a} = 2^{b-1} - 1$.

In this scheme, λ is a variable to control the distribution smoothness of $\tilde{\mathbf{W}}$ between -1 and 1. We depict a series of $\hat{\mathbf{W}}$ as well as the corresponding quantization error $\mathbf{J}$ and information entropy $\mathbf{E}$ of quantized weights $\mathbf{Q}$ as λ changes among $\{1,2,3,4,5\}$. Suggested by (Qin et al. 2020), $\mathbf{J}$ and $\mathbf{E}$ are defined as:

$$\begin{aligned} \mathbf{J} &= ||\mathbf{W} - \mathbf{Q}||^2, \\ \mathbf{E} &= -\sum_{x\in\Omega} \mathbf{q}(x) log(\mathbf{q}(x)) \end{aligned} \tag{7}$$

where $\mathbf{q}(x)$ is the probability mass function of quantized weights $\mathbf{Q}$ and Ω denotes its value range. As shown in Fig 4(a), the re-scaling operation makes $\hat{\mathbf{W}}$ distributed more uniform, which is beneficial for quantization. As λ increases, $\hat{\mathbf{W}}$ becomes more centralized, and the quantization error as well as information entropy decreases simultaneously. However, as indicated by (Qin et al. 2020), it is essential to minimize the quantization error as well as maximize the information entropy for a better quantization performance. This suggests that we must make a good trade-off between these two terms by adjusting λ. Since it is difficult to adjust λ manually, we make it trainable, learning along with other model parameters to seek the optimal solution. Compared with DoReFa scheme, the introduction of λ makes ASQ be able to adaptively control the distribution of $\mathbf{Q}$ for keeping a balance between quantization error and information entropy for each layer. With 4-bit ASQ, EtinyNet can be further compressed by half with less accuracy drop.

Experiments

In this section, we firstly compare EtinyNet with other state-of-the-art lightweight CNN models on ImageNet dataset to demonstrate its extremely high parameter efficiency. Then, the superior results of deploying EtinyNet on IoT devices are discussed for verifying its effectiveness in practical TinyML.

Classification Results on ImageNet

Training details. ImageNet-1000 (Deng et al. 2009) is the most convincing benchmark which consists of 1,281,167 images belonging 1000 categories. We conduct extensive experiments on ImageNet using the MXNet (Chen et al. 2015) toolbox. During training, we use the standard SGD optimizer to train our models with both decay and momentum of 0.9 and the weight decay is $1e-4$. We use the cosine learning schedule with an initial learning rate of 0.1 and the weight initialization introduced by (He et al. 2015). The batch size is set to 1024 and 8 GPUs are used for training. We train all the models for 300 epochs. The input image is randomly cropped to 224 $\times$ 224 and randomly flipped horizontally, and is kept as 8 bit signed integer with no standardization applied. The standard data augmentations provided by MXNet are used and there is no other regularization, *e.g.* mixup, label smoothing, in the training.

When training quantized models using 4-bit ASQ, we adopt the full-precision models with clamped weights as initial points. All convolution layers and fully-connected layers are quantized. The initial learning rate and training epochs are adjusted to 0.01 and 40. We initialize λ as 2. Other settings are the same as that of training full-precision models.

Accuracy. Table 2 lists the results of the most well-known lightweight CNN architectures, including MobileNet series, ShuffleNet series, and EfficientNet, *etc*, which have weights between 1M and 2M, or an equivalent full-precision model size of 4MB to 8MB. Among all these results, our EtinyNet-1.0 achieves the highest accuracy, reaching 65.5% top-1 accuracy and 86.2% top-5 accuracy. It outperforms the current best result achieved by MobileNeXt-0.35 with significant 0.8%, and is 1.1% higher than the second highest result obtained by MnasNet-A1-0.35. EtinyNet-0.75, a more compact version, obtains a relatively lower top-1 accuracy of 62.2%. However, it is still superior to most of the other models only except for MobileNeXt-0.35 and MnasNet-A1-0.35. Furthermore, it's worth noting that even though the capacity of EtinyNet is especially small, the 4-bit quantized

Model	Params. (MB)	MAdds (M)	Top-1 Acc. (%)	Top-5 Acc. (%)
MobileNeXt-0.35 (Daquan et al. 2020)	7.2	80	64.7	-
MnasNet-A1-0.35 (Tan et al. 2019)	6.8	63	64.4	85.1
MobileNetV2-0.35 (Sandler et al. 2018)	6.8	59	60.3	82.9
MicroNet-M3 (Li et al. 2021)	6.4	20	61.3	82.9
MobileNetV3-Small-0.35 (Howard et al. 2019)	6.4	23	58.0	-
ShuffleNetV2-0.5 (Ma et al. 2018)	5.6	41	61.1	82.6
MicroNet-M2 (Li et al. 2021)	5.6	11	58.2	80.1
MobileNetV2-0.15 (Sandler et al. 2018)	5.6	39	55.1	-
MobileNetV1-0.5 (Howard et al. 2017)	5.2	110	61.7	83.6
EfficientNet-B (Tan and Le 2019)	5.2	24	56.7	79.8
EtinyNet-1.0	3.92	117	**65.5**	**86.2**
EtinyNet-0.75	2.72	75	62.2	84.0
EtinyNet-1.0 (4-bit)	0.49	117	60.7	82.8
EtinyNet-0.75 (4-bit)	**0.34**	75	57.0	80.2

Table 2: Comparison of state-of-the-art small networks over classification accuracy, the model size and MAdds on ImageNet-1000 dataset. "-" mean no reported results available.

version can further shrink model size while maintain comparable accuracy, resulting 60.7% for EtinyNet-1.0 and 57.0 for EtinyNet-0.75.

Parameter Efficiency. Obviously, the proposed EtinyNet series have the smallest model size of all the architectures listed in Table 2. Even the largest model EtinyNet-1.0 is just 3.92MB (0.98M parameters), about 54% the size of MobileNeXt-0.35. With the higher accuracy, EtinyNet-0.75 only has half the parameters of MobileNetV1-0.5, which further demonstrates the excellent parameter efficiency of EtinyNet. Moreover, by applying 4-bit ASQ, the size of the EtinyNet-0.75 can be further reduced to just 340KB, about one-fifteenth the size of EfficientNet-B. To the best of our knowledge, this is the minimal CNN model that achieves $>$ 55% ImageNet top-1 accuracy, bringing the ImageNet-scale CNN models to an unprecedented tiny size. The high parameter efficiency of our EtinyNet introduces two obvious benefits: 1) enabling better accuracy of running CNNs on IoT devices; 2) reducing the power consumption caused by the transmission of mode parameters. In the following section, we will experimentally verify these two benefits on IoT devices.

Effectiveness of the proposed building blocks. To verify the effectiveness of the proposed LB and DLB, we substitute all the blocks in EtinyNet with the widely applied inverted bottleneck block that is used in MobileNetV2 and EfficientNet, *etc*, while keeping a similar number of parameters. Experimental results show that it would bring about up to 2.4% accuracy drop, which indicates that the proposed LB and DLB does have an edge over existing efficient block structurally. In addition, we find that adding non-linear activations (ReLU) between depthwise and pointwise convolutional layer decreases the performance by nearly 0.8% compared to the setting of removing ReLU. This indicates that linear depthwise is also essential for lightweight networks with high parameter efficiency.

Effectiveness of ASQ. We further demonstrate that the proposed ASQ is more effective than DoReFa scheme for quantization on tiny models by comparing the top-1 accuracy of 4-bit quantized EtinyNet on ImageNet-1000. As shown in Table 3, the accuracy gap between two methods on EtinyNet-1.0 is 5.6%. When applied on the smaller model EtinyNet-0.75, the accuracy gap is even widened to 11.7%, which suggests that DoReFa degrades seriously as model size decreases while the proposed ASQ is more suitable for tiny models.

Methods	EtinyNet-1.0	EtinyNet-0.75
ASQ	60.7%	57.0%
DoReFa	55.1%	45.3%

Table 3: Accuracy of 4-bit quantized EtinyNet on ImageNet-1000 using ASQ and DoReFa.

Methods	STM32F412 (256KB 1MB)	STM32F746 (320KB 1MB)
Rusci *et al.*	60.2%	-
MCUNet	62.2%	63.5%
EtinyNet-1.0	64.7%	65.8%

Table 4: Comparison to MCU designs on ImageNet-1000 for object classification. EtinyNet obtains the record accuracy of 64.7% and 65.8% on STM32F412 and STM32F746.

Results on IoT devices

Comparisons to MCU-based designs. The current mainstream solution for TinyML is to run CNNs on MCUs, as suggested by (Lin et al. 2020; Sudharsan, Breslin, and Ali 2021; Banbury et al. 2020). Following these works, we deploy EtinyNet-1.0 for ImageNet classification on STM32F412 (ARM Cortex-M4 CPU with 256KB SRAM and 1MB Flash) and STM32F746 MCUs. We quantize EtinyNet-1.0 using 8-bit for both weights and activations, and the size of input image is set as 224 and 256 on

Methods	MAdds	Params.	Peak SRAM	mAP
CMSIS	34M	0.87MB	519KB	31.6%
MCUNet	168M	1.2MB	466KB	51.4%
EtinyNet-SSD	164M	0.59MB	395KB	56.4%

Table 5: Comparisons to MCU designs on Pascal VOC for object detection. EtinyNet-SSD improves the detection mAP by 5% on STM32H743. We are able to fit a model with much less MAdds, parameters and peak SRAM consumption.

STM32F412 and STM32F746. The results are shown in Table 4. EtinyNet outperforms the state-of-the-art MCUNet and Rusci *et al.* (Rusci, Capotondi, and Benini 2020) with a big margin on both MCUs, reaching the record accuracy of 64.7% and 65.8%, respectively. The superior performances on MCUs suggest the availability of the proposed EtinyNet architecture in practical TinyML.

To show the generality of EtinyNet architecture across different tasks, we also apply the EtinyNet to the task of object detection. We take the EtinyNet-1.0 as backbone to extract features and build the detection layers as well as prior boxes presented by SSD (Liu et al. 2016) on the head of EtinyNet-1.0. The resulting network is denoted as EtinyNet-SSD. We adopt 256 $\times$ 256 input images as input for better detection of small objects, and use 8-bit linear quantization for weights and activations. The object detection performance of our EtinyNet-SSD and other state-of-the-art methods are benchmarked on Pascal VOC (Everingham et al. 2009) dataset using STM32H743 (ARM Cortex-M7 CPU with 512KB SRAM and 2MB Flash) MCU. As shown in Table 5, EtinyNet-SSD achieves highest mAP of 56.4% with less MAdds and peak SRAM consumption, and only half the parameters of MCUNet, showing its prominent parameter efficiency. Less storage consumption and higher performance of EtinyNet-SSD makes AIoT applications more accessible.

Comparisons to FPGA-based designs. FPGAs are frequently utilized for accelerating CNNs inference thanks to its parallel computing capability. However, existing FPGA designs (Lian et al. 2019; Xiao et al.; Guo et al. 2018; Yu et al. 2020) generally consume several watts of power due to the large size of CNN models, making it difficult to be applied to the TinyML scenarios that require very low energy consumption. We performe experiments on the Xilinx compact FPGA Artix7 XC7A100T to demonstrate the feasibility of achieving milliwatts of power for running tiny model. Since there is only 607.5KB on-chip storage in XC7A100T, we deploy the smallest model EtinyNet-0.75 (4-bit) with 8-bit quantization for activations. Table 6 exhibits our results as well as the comparisons to AngelEye (Guo et al. 2018) and SSA (Shaydyuk and John 2020). Ii is clear that our design achieves the lowest power of only 0.62W, about 5.6 $\times$ lower than AngelEye. Different from other designs using off-chip DRAM to cache feature maps and model parameters, we are able to complete all computation with only on-chip SRAM, profited from the compactness of our tiny model. As a result, we save much energy consumption caused by off-chip memory access. As far as we know, we are the first to achieve milliwatts of power with $>$ 55% ImageNet top-1 accuracy on FPGAs. More than that, less parameters reduce the latency of data movement and thus achieving higher throughput of 103FPS. In summary, deploying the proposed EtinyNet on FPGA achieves roughly 10 $\times$ processing efficiency gains (17.1 Frames/s/mJ v.s. 1.6 Frames/s/mJ), which provides a powerful solution for TinyML.

Component	AngelEye	SSA	Ours
Device	ZYNQ XCZ7020	ZYNQ XCZU7EV	Artix7 XC7A100T
On-chip RAM used (KB)	385	1483	576
DRAM used	yes	yes	**none**
ImageNet accuracy (%)	67.72	-	56.0
Model params. (MB)	138	3.5	0.34
Processing time (ms)	354	10.1	**9.7**
Power (W)	3.5	6.2	**0.62**
Efficiency (Frames/s/mJ)	0.0023	1.6	**17.1**

Table 6: Comparisons to other state-of-the-art FPGA designs. By deploying the smallest EtinyNet-0.75 (4bit), our FPGA design achieves competitive throughput and reduction of power by 5.6 $\times$.

Conclusion

In this paper, we aim at developing the efficient tiny model with only hundreds of KB parameters to meet very limited storage and power requirements of TinyML scenarios. Toward this end, we firstly design a parameter-efficient tiny architecture by introducing dense linear depthwise block. Then, a novel adaptive scale quantization (ASQ) method is proposed for further quantizing tiny models in aggressive low-bit while retaining the accuracy. With the optimized architecture and 4-bit ASQ, we achieve a family of networks, named EtinyNet, that yields 57.0% ImageNet top-1 accuracy with an unprecedented model size of only 340KB. We empirically demonstrate that the it can achieve much higher processing efficiency on kinds of IoT devices with the proposed EtinyNet due to its real compactness. The proposed extremely tiny models make it possible to run CNNs in a single chip, showing the potential for designing small footprint ASIC CNNs accelerator of more higher efficiency.

Acknowledgements

This work was supported by National Key R&D Program of China(No.2018YF70202800), the Natural Science Foundation of China (No.61674120), JST, ACT-X (No.JPMJAX190D), Japan and JST Moonshot R&D

(No.JPMJMS2011), Fundamental Research Funds for Central Universities and Innovation Fund of Xidian University.

References

Bai, Y.; Wang, Y.-X.; and Liberty, E. 2019. ProxQuant: Quantized Neural Networks via Proximal Operators. *ArXiv*, abs/1810.00861.

Banbury, C. R.; Zhou, C.; Fedorov, I.; Navarro, R. M.; Thakker, U.; Gope, D.; Reddi, V. J.; Mattina, M.; and Whatmough, P. N. 2020. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *CoRR*, abs/2010.11267.

Bengio, Y.; Léonard, N.; and Courville, A. C. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *ArXiv*, abs/1308.3432.

Capotondi, A.; Rusci, M.; Fariselli, M.; and Benini, L. 2020. CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices. *IEEE Trans. Circuits Syst. II Express Briefs*, 67-II(5): 871–875.

Chappa, R. T. N.; and El-Sharkawy, M. 2020. Deployment of SE-SqueezeNext on NXP BlueBox 2.0 and NXP i.MX RT1060 MCU. In *2020 IEEE Midwest Industry Conference (MIC)*, volume 1, 1–4.

Chen, T.; Li, M.; Li, Y.; Lin, M.; Wang, N.; Wang, M.; Xiao, T.; Xu, B.; Zhang, C.; and Zhang, Z. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR*, abs/1512.01274.

Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E. Q.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; Guestrin, C.; and Krishnamurthy, A. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *OSDI*.

Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P. I.; Srinivasan, V.; and Gopalakrishnan, K. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *CoRR*, abs/1805.06085.

Daquan, Z.; Hou, Q.; Chen, Y.; Feng, J.; and Yan, S. 2020. Rethinking Bottleneck Structure for Efficient Mobile Network Design. *ArXiv*, abs/2007.02269.

David, R.; Duke, J.; Jain, A.; Reddi, V. J.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Regev, S.; Rhodes, R.; Wang, T.; and Warden, P. 2020. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. *CoRR*, abs/2010.08678.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*.

Everingham, M.; Gool, L.; Williams, C. K. I.; Winn, J.; and Zisserman, A. 2009. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88: 303–338.

Gong, R.; Liu, X.; Jiang, S.; Li, T.; Hu, P.; Lin, J.; Yu, F.; and Yan, J. 2019. Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 4851–4860.

Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; and Yang, H. 2018. Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(1): 35–47.

Han, K.; Wang, Y.; Zhang, Q.; Zhang, W.; Xu, C.; and Zhang, T. 2020. Model Rubik's Cube: Twisting Resolution, Depth and Width for TinyNets. *ArXiv*, abs/2010.14819.

Han, S.; Mao, H.; and Dally, W. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *arXiv: Computer Vision and Pattern Recognition*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1026–1034.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Howard, A. G.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; Le, Q. V.; and Adam, H. 2019. Searching for MobileNetV3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 1314–1324.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv*, abs/1704.04861.

Hu, J.; Shen, L.; Albanie, S.; Sun, G.; and Wu, E. 2020. Squeeze-and-Excitation Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42: 2011–2023.

Huang, G.; Liu, Z.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261–2269.

Jin, Q.; Yang, L.; and Liao, Z. A. 2019. Towards Efficient Training for Neural Network Quantization. *ArXiv*, abs/1912.10207.

Jin, Q.; Yang, L.; and Liao, Z. A. 2020. AdaBits: Neural Network Quantization With Adaptive Bit-Widths. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2143–2153.

Lai, L.; Suda, N.; and Chandra, V. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *CoRR*, abs/1801.06601.

Lee, H.; Battle, A.; Raina, R.; and Ng, A. 2006. Efficient sparse coding algorithms. In *NIPS*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. 2017. Pruning Filters for Efficient ConvNets. *ArXiv*, abs/1608.08710.

Li, Y.; Chen, Y.; Dai, X.; Chen, D.; Liu, M.; Yuan, L.; Liu, Z.; Zhang, L.; and Vasconcelos, N. 2021. MicroNet: Improving Image Recognition with Extremely Low FLOPs. *ArXiv*, abs/2108.05894.

Lian, X.; Liu, Z.; Song, Z.; Dai, J.; Zhou, W.; and Ji, X. 2019. High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic. *IEEE Trans. Very Large Scale Integr. Syst.*, 27(8): 1874–1885.

Lin, J.; Chen, W.; Lin, Y.; Cohn, J.; Gan, C.; and Han, S. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. *ArXiv*, abs/1806.09055.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S. E.; Fu, C.-Y.; and Berg, A. 2016. SSD: Single Shot MultiBox Detector. In *ECCV*.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *2017 IEEE International Conference on Computer Vision (ICCV)*, 5068–5076.

Ma, N.; Zhang, X.; Zheng, H.; and Sun, J. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *ArXiv*, abs/1807.11164.

Qin, H.; Gong, R.; Liu, X.; Shen, M.; Wei, Z.; Yu, F.; and Song, J. 2020. Forward and Backward Information Retention for Accurate Binary Neural Networks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2247–2256.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*.

Rusci, M.; Capotondi, A.; and Benini, L. 2020. Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers. *ArXiv*, abs/1905.13082.

Sandler, M.; Howard, A. G.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Shaydyuk, N. K.; and John, E. 2020. FPGA Implementation of MobileNetV2 CNN Model Using Semi-Streaming Architecture for Low Power Inference Applications. *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 160–167.

Sudharsan, B.; Breslin, J. G.; and Ali, M. I. 2021. ML-MCU: A Framework to Train ML Classifiers on MCU-based IoT Edge Devices. *IEEE Internet of Things Journal*, 1–1.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; and Le, Q. V. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2815–2823.

Tan, M.; and Le, Q. V. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv*, abs/1905.11946.

Tang, Y.; You, S.; Xu, C.; Han, J.; Qian, C.; Shi, B.; Xu, C.; and Zhang, C. 2020. Reborn Filters: Pruning Convolutional Neural Networks with Limited Data. In *AAAI*.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8604–8612.

Wu, Z.; Shen, C.; and Hengel, A. V. 2019. Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. *Pattern Recognit.*, 90: 119–133.

Xiao, Q.; Liang, Y.; Lu, L.; Yan, S.; and Tai, Y. ???? Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference, Austin, TX, USA, June 18-22, 2017*, 62:1–62:6. ACM.

Yu, Y.; Wu, C.; Zhao, T.; Wang, K.; and He, L. 2020. OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(1): 35–47.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. *ArXiv*, abs/1605.07146.

Zhang, D.; Yang, J.; Ye, D.; and Hua, G. 2018. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. *ArXiv*, abs/1807.10029.

Zhou, S.; Ni, Z.; Zhou, X.; Wen, H.; Wu, Y.; and Zou, Y. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR*, abs/1606.06160.

Zim, M. Z. H. 2021. TinyML: Analysis of Xtensa LX6 microprocessor for Neural Network Applications by ESP32 SoC. *CoRR*, abs/2106.10652.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. *ArXiv*, abs/1611.01578.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning Transferable Architectures for Scalable Image Recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8697–8710.