MobileOne: An Improved One millisecond Mobile Backbone

Pavan Kumar Anasosalu Vasu James Gabriel Jeff Zhu Oncel Tuzel Anurag Ranjan

Apple

Abstract

Efficient neural network backbones for mobile devices are often optimized for metrics such as FLOPs or parameter count. However, these metrics may not correlate well with latency of the network when deployed on a mobile device. Therefore, we perform extensive analysis of different metrics by deploying several mobile-friendly networks on a mobile device. We identify and analyze architectural and optimization bottlenecks in recent efficient neural networks and provide ways to mitigate these bottlenecks. To this end, we design an efficient backbone MobileOne, with variants achieving an inference time under 1 ms on an iPhone12 with 75.9% top-1 accuracy on ImageNet. We show that MobileOne achieves state-of-the-art performance within the efficient architectures while being many times faster on mobile. Our best model obtains similar performance on ImageNet as MobileFormer while being 38× faster. Our model obtains 2.3% better top-1 accuracy on ImageNet than EfficientNet at similar latency. Furthermore, we show that our model generalizes to multiple tasks – image classification, object detection, and semantic segmentation with significant improvements in latency and accuracy as compared to existing efficient architectures when deployed on a mobile device. Code and models are available at `https://github.com/apple/ml-mobileone`

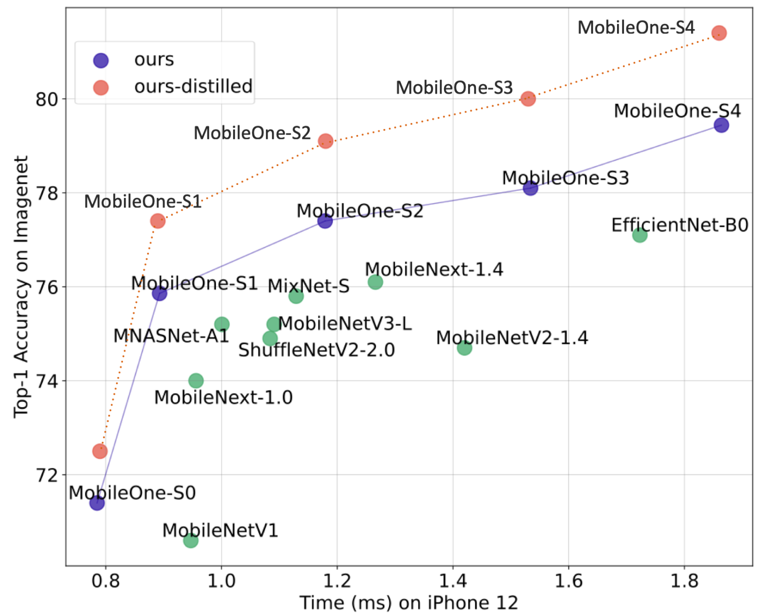
1. Introduction

Design and deployment of efficient deep learning architectures for mobile devices has seen a lot of progress [5, 30, 31, 42, 44, 46] with consistently decreasing floating-point operations (FLOPs) and parameter count while improving accuracy. However, these metrics may not correlate well with the efficiency [9] of the models in terms of latency. Efficiency metric like FLOPs do not account for memory access cost and degree of parallelism, which can have a non-trivial effect on latency during inference [42]. Parameter count is also not well correlated with latency. For example, sharing parameters leads to higher FLOPS but smaller model size. Furthermore, parameter-less operations like skip-connections [24] or branching [33, 49] can incur significant memory access costs. This disconnect can get exacerbated when custom accelerators are available in the regime of efficient architectures.

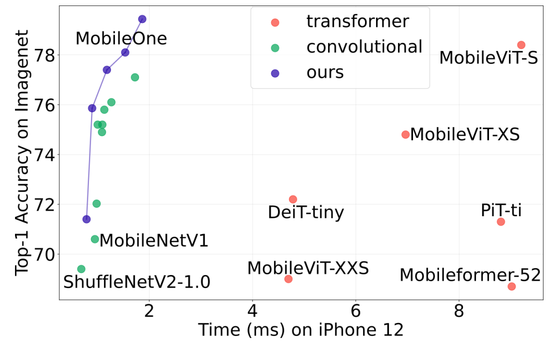
Our goal is to improve the latency cost of efficient architectures while improving their accuracy by identifying key architectural and optimization bottlenecks that affect on-device latency. To identify architectural bottlenecks, we deploy neural networks on an iPhone12 by using CoreML [56] and benchmark their latency costs. To alleviate optimization bottlenecks, we decouple train-time and inference-time architectures, i.e. using a linearly over-parameterized model at train-time and re-parameterizing the linear structures at inference [11–13]. We further alleviate optimization bottleneck by dynamically relaxing regularization throughout training to prevent the already small models from being over-regularized.

Based on our findings on the key bottlenecks, we design a novel architecture *MobileOne*, variants of which run under 1 ms on an iPhone12 achieving state-of-the-art accuracy within efficient architecture family while being significantly faster on the device. Like prior works on structural re-parameterization [11–13], MobileOne introduces linear branches at train-time which get re-parameterized at inference. However, a key difference between our model and prior structural re-parameterization works is the introduction of trivial over-parameterization branches, which provides further improvements in low parameter regime and model scaling strategy. At inference, our model has simple feed-forward structure without any branches or skip-connections. Since this structure incurs lower memory access cost, we can incorporate wider layers in our network which boosts representation capacity as demonstrated empirically in Table 9. For example, MobileOne-S1 has 4.8M parameters and incurs a latency of 0.89ms, while MobileNet-V2 [46] has 3.4M (29.2% less than MobileOne-S1) parameters and incurs a latency of 0.98ms. At this operating point, MobileOne attains 3.9% better top-1 accuracy than MobileNet-V2.

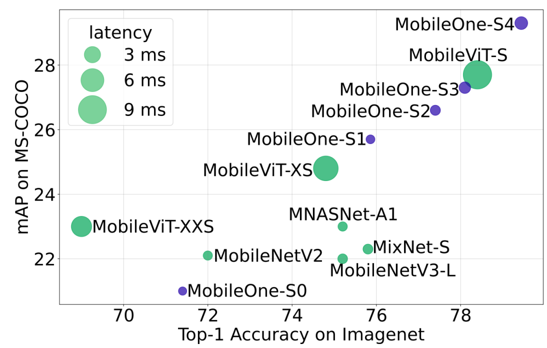
MobileOne achieves significant improvements in latency compared to efficient models in literature while maintain-



(a) Top 1 accuracy vs Latency on iPhone 12.



(b) Zoomed out (a)



(c) Top-1 accuracy vs mAP.

Figure 1. We show comparisons of Top-1 accuracy on image classification vs latency on an iPhone 12 (a), and zoomed out area (b) to include recent transformer architectures. We show mAP on object detection vs Top-1 accuracy on image classification in (c) with size of the marker indicating latency of the backbone on iPhone 12. Our models have significantly smaller latency compared to related works. Please refer to supp. mat. for higher resolution figures.

ing the accuracy on several tasks – image classification, object detection, and semantic segmentation. As shown in Figure 1b, MobileOne performs better than MobileViT-S [44] while being 5 $\times$ faster on image classification. As compared to EfficientNet-B0 [53], we achieve 2.3% better top-1 accuracy on ImageNet [10] with similar latency costs (see Figure 1a). Furthermore, as seen in Figure 1c, MobileOne models not only perform well on ImageNet, they also generalize to other tasks like object detection. Models like MobileNetV3-L [30] and MixNet-S [54] improve over MobileNetV2 on ImageNet, but those improvements do not translate to object detection task. As shown in Figure 1c, MobileOne shows better generalization across tasks. For object detection on MS-COCO [37], best variant of MobileOne outperforms best variant MobileViT by 6.1% and MNASNet by 27.8%. For semantic segmentation, on PascalVOC [16] dataset, best variant of MobileOne outperforms best variant MobileViT by 1.3% and on ADE20K [64] dataset, best variant of MobileOne outperforms MobileNetV2 by 12.0%. In summary, our contributions are as follows:

- We introduce *MobileOne*, a novel architecture that runs within 1 ms on a mobile device and achieves state-of-the-art accuracy on image classification within efficient model architectures. The performance of our model also generalizes to a desktop CPU and GPU.
- We analyze performance bottlenecks in activations and branching that incur high latency costs on mobile in recent efficient networks.
- We analyze the effects of train-time re-parameterizable branches and dynamic relaxation of regularization in training. In combination, they help alleviating optimization bottlenecks encountered when training small models.
- We show that our model generalizes well to other tasks – object detection and semantic segmentation while outperforming recent state-of-the-art efficient models.

We will release our trained networks and code for research purposes. We will also release the code for iOS application to enable benchmarking of networks on iPhone.

2. Related Work

Designing a real-time efficient neural network involves a trade-off between accuracy and performance. Earlier methods like SqueezeNet [34] and more recently MobileViT [44], optimize for parameter count and a vast majority of methods like MobileNets [31, 46], MobileNeXt [65], ShuffleNet-V1 [63], GhostNet [20], MixNet [54] focus on optimizing for the number of floating-point operations (FLOPs). EfficientNet [53] and TinyNet [21] study the compound scaling of depth, width and resolution while optimizing FLOPs. Few methods like MNASNet [52], MobileNetV3 [30] and ShuffleNet-V2 [42] optimize directly for latency. Dehghani et al. [9] show that FLOPs and parameter count are not well correlated with latency. Therefore, our work focuses on improving on-device latency while improving the accuracy.

Recently, ViT [14] and ViT-like architectures [57] have shown state-of-the-art performance on ImageNet dataset. Different designs like ViT-C [61], CvT [60], BoTNet [48], ConViT [8] and PiT [29] have been explored to incorporate biases using convolutions in ViT. More recently, MobileFormer [5] and MobileViT [44] were introduced to get ViT-like performance on a mobile platform. MobileViT optimizes for parameter count and MobileFormer optimizes for FLOPs and outperforms efficient CNNs in low FLOP regime. However, as we show in subsequent sections that low FLOPs does not necessarily result in low latency. We study key design choices made by these methods and their impact on latency.

Recent methods also introduce new architecture designs and custom layers to improve accuracy for mobile backbones. MobileNet-V3 [30], introduces an optimized activation function – Hard-Swish for a specific platform. However, scaling such functions to different platforms may be difficult.

Therefore, our design uses basic operators that are already available across different platforms. ExpandNets [19], ACNet [11] and DBBNet [12], propose a drop-in replacement for a regular convolution layer in recent CNN architectures and show improvements in accuracy. RepVGG [13] introduces re-parameterizable skip connections which is beneficial to train VGG-like model to better performance. These architectures have linear branches at train-time that get re-parameterized to simpler blocks at inference. We build on these re-parametrization works and introduce trivial over-parameterization branches thereby providing further improvements in accuracy.

3. Method

In this section, we analyse the correlation of popular metrics – FLOPs and parameter count – with latency on a mobile device. We also evaluate how different design choices in architectures effect the latency on the phone. Based on the evaluation, we describe our architecture and training algorithm.

Type	FLOPs corr.	FLOPs p-value	Parameters corr.	Parameters p-value
Mobile Latency	0.47	0.03	0.30	0.18
CPU Latency	0.06	0.80	0.07	0.77

Table 1. Spearman rank correlation coeff. between latency-flops.

3.1. Metric Correlations

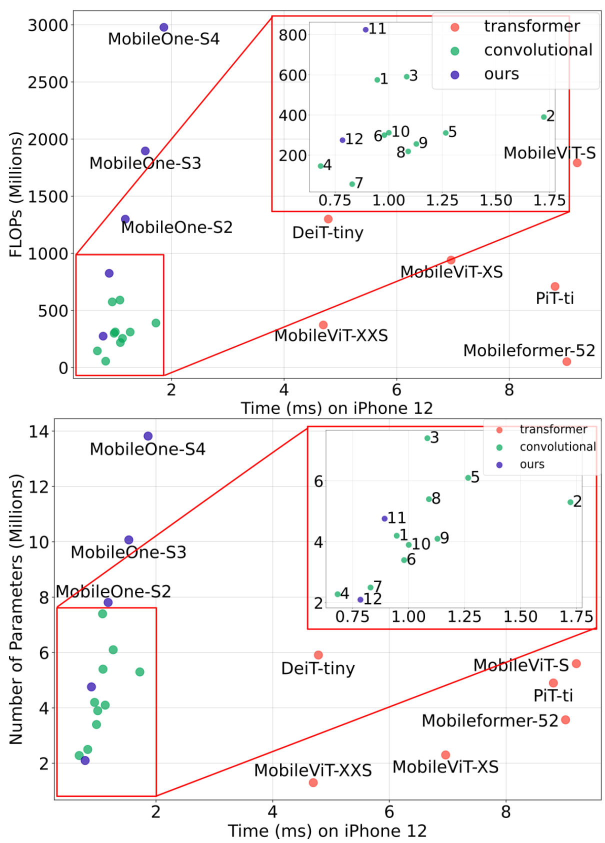
The most commonly used cost indicators for comparing the size of two or more models are parameter count and FLOPs [9]. However, they may not be well correlated with latency in real-world mobile applications. Therefore, we study the correlation of latency with FLOPS and parameter count for benchmarking efficient neural networks. We consider recent models and use their Pytorch implementation to convert them into ONNX format [2]. We convert each of these models to coreml packages using Core ML Tools [56]. We then develop an iOS application to measure the latency of the models on an iPhone12.

We plot latency vs. FLOPs and latency vs. parameter count as shown in Figure 2. We observe that many models with higher parameter count can have lower latency. We observe a similar plot between FLOPs and latency. Furthermore, we note the convolutional models such as MobileNets [42, 46, 55] have lower latency for similar FLOPs and parameter count than their transformer counterparts [5, 44, 57]. We also estimate the Spearman rank correlation [62] in Table 1a. We find that latency is moderately correlated with FLOPs and weakly correlated with parameter counts for efficient architectures on a mobile device. This correlation is even lower on a desktop CPU.

3.2. Key Bottlenecks

Activation Functions To analyze the effect of activation functions on latency, we construct a 30 layer convolutional neural network and benchmark it on iPhone12 using different activation functions, commonly used in efficient CNN backbones. All models in Table 2 have the same architecture except for activations, but their latencies are drastically different. This can be attributed to synchronization costs mostly incurred by recently introduced activation functions like SE-ReLU [32], Dynamic Shift-Max [36] and DynamicReLUs [6]. DynamicReLU and Dynamic Shift-Max have shown significant accuracy improvement in extremely low FLOP models like MicroNet [36], but, the latency cost of using these activations can be significant. Therefore we use only ReLU activations in MobileOne.

Architectural Blocks Two of the key factors that affect runtime performance are memory access cost and degree of parallelism [42]. Memory access cost increases significantly in multi-branch architectures as activations from each branch have to be stored to compute the next tensor



1 MobileNetV1	2 EfficientNet-B0	3 ShuffleNetV2-2.0	4 ShuffleNetV2-1.0
5 MobileNext-1.4	6 MobileNetV2	7 MobileNetV3-S	8 MobileNetV3-L
9 MixNet-S	10 MNASNet-A1	11 MobileOne-S1	12 MobileOne-S0

Figure 2. Top: FLOPs vs Latency on iPhone12. Bottom: Parameter Count vs Latency on iPhone 12. We indicate some networks using numbers as shown in the table above.

Activation Function	Latency (ms)
ReLU [1]	1.53
GELU [27]	1.63
SE-ReLU [32]	2.10
SiLU [15]	2.54
Dynamic Shift-Max [36]	57.04
DynamicReLU-A [6]	273.49
DynamicReLU-B [6]	242.14

Table 2. Comparison of latency on mobile device of different activation functions in a 30-layer convolutional neural network.

in the graph. Such memory bottlenecks can be avoided if the network has smaller number of branches. Architectural blocks that force synchronization like global pooling operations used in Squeeze-Excite block [32] also affect overall run-time due to synchronization costs. To demonstrate the hidden costs like memory access cost and synchronization cost, we ablate over using skip connections and squeeze-excite blocks in a 30 layer convolutional neural network. In Table 3b, we show how each of these choices contribute towards latency. Therefore we adopt an architecture with no branches at inference, which results in smaller memory access cost. In addition, we limit the use of Squeeze-Excite blocks to our biggest variant in order to improve accuracy.

Architectural Blocks	Baseline	+ Squeeze Excite [32]	+ Skip Connections [23]
Latency (ms)	1.53	2.10	2.62

Table 3. Ablation on latency of different architectural blocks in a 30-layer convolutional neural network.

3.3. MobileOne Architecture

Based on the our evaluations of different design choices, we develop the architecture of MobileOne. Like prior works on structural re-parameterization [11–13, 19], the train-time and inference time architecture of MobileOne is different. In this section, we introduce the basic block of MobileOne and the model scaling strategy used to build the network.

MobileOne Block MobileOne blocks are similar to blocks introduced in [11–13, 19], except that our blocks are designed for convolutional layers that are factorized into depthwise and pointwise layers. Furthermore, we introduce trivial over-parameterization branches which provide further accuracy gains. Our basic block builds on the MobileNet-V1 [31] block of 3x3 depthwise convolution followed by 1x1 pointwise convolutions. We then introduce reparameterizable skip connection [13] with batchnorm along with branches that replicate the structure as shown in Figure 3. The trivial over-parameterization factor k is a hyperparameter which is varied from 1 to 5. We ablate over the choice for k in Table 4. At inference, MobileOne model does not have any branches. They are removed using the re-parameterization process described in [12, 13].

For a convolutional layer of kernel size K, input channel dimension C_{in} and output channel dimension C_{out}, the weight matrix is denoted as $\mathbf{W}' \in \mathbb{R}^{C_{out} \times C_{in} \times K \times K}$ and bias is denoted as $\mathbf{b}' \in \mathbb{R}^{D}$. A batchnorm layer contains accumulated mean $\boldsymbol{\mu}$, accumulated standard deviation $\boldsymbol{\sigma}$, scale $\boldsymbol{\gamma}$ and bias $\boldsymbol{\beta}$. Since convolution and batchnorm at inference are linear operations, they can be folded into a single convolution layer with weights $\widehat{\mathbf{W}} = \mathbf{W}' * \frac{\boldsymbol{\gamma}}{\boldsymbol{\sigma}}$ and bias $\widehat{\mathbf{b}} = (\mathbf{b}' - \boldsymbol{\mu}) * \frac{\boldsymbol{\gamma}}{\boldsymbol{\sigma}} + \boldsymbol{\beta}$. Batchnorm is folded into preceding convolutional layer in all the branches. For skip connection the batchnorm is folded to a convolutional layer with identity 1x1 kernel, which is then padded by $K - 1$ zeros as described in [13]. After obtaining the batchnorm folded weights in each branch, the weights $\mathbf{W} = \sum_i^M \widehat{\mathbf{W}}_i$ and bias $\mathbf{b} = \sum_i^M \widehat{\mathbf{b}}_i$ for convolution layer at inference is

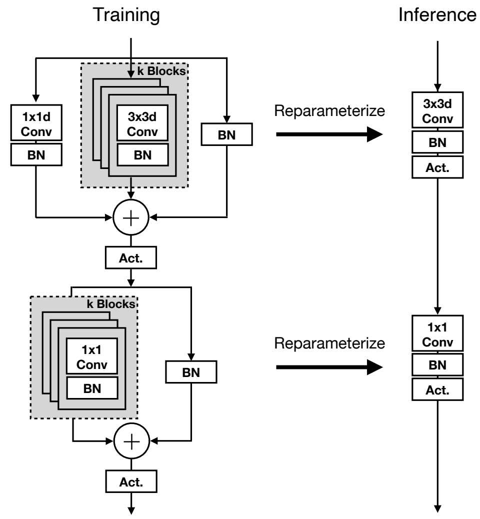


Figure 3. MobileOne block has two different structures at train time and test time. Left: Train time MobileOne block with reparameterizable branches. Right: MobileOne block at inference where the branches are reparameterized. Either ReLU or SE-ReLU is used as activation. The trivial over-parameterization factor k is a hyperparameter which is tuned for every variant.

Model	# Params.	Top-1
ExpandNet-CL MobileNetV1 [19]	4.2	69.4
RepVGG-A0 [13]	8.3	72.4
RepVGG-A1 [13]	12.8	74.5
RepVGG-B0 [13]	14.3	75.1
ACNet MobileNetV1 [11]	4.2	72.1
ACNet ResNet18 [11]	11.7	71.1
DBBNet MobileNetV1 [12]	4.2	72.9
DBBNet ResNet18 [12]	11.7	71.0
MobileOne-S0	2.1	71.4
MobileOne-S1	4.8	75.9
MobileOne-S2	7.8	77.4
MobileOne-S3	10.1	78.1
MobileOne-S4	14.8	79.4

Table 4. Comparison of Top-1 Accuracy on ImageNet against recent train time over-parameterization works. Number of parameters listed above is at inference.

Re-param.	MobileOne-S0	MobileOne-S1	MobileOne-S3
with	71.4	75.9	78.1
without	69.6	74.6	77.2

Table 5. Effect re-parametrizable branches on Top-1 ImageNet accuracy.

obtained, where M is the number of branches.

To better understand the improvements from using train time re-parameterizable branches, we ablate over versions of MobileOne models by removing train-time re-parameterizable branches (see Table 5), while keeping all other training parameters the same as described in Section 4. Using re-parameterizable branches significantly improves performance. To understand the importance of trivial over-parameterization branches, we ablate over the choice of over-parameterization factor k in Table 6. For larger variants of MobileOne, the improvements from trivial over-parameterization starts diminishing. For smaller variant like MobileOne-S0, we see improvements of 0.5% by using trivial over-parameterization branches. In Figure 4, we see that adding re-parameterizable branches improves optimization as both train and validation losses are further lowered.

Model	Top-1				
	k=1	k=2	k=3	k=4	k-5
MobileOne-S0	70.9	70.7	71.3	71.4	71.1
MobileOne-S1	75.9	75.7	75.6	75.6	75.2

Table 6. Comparison of Top-1 on ImageNet for various values of trivial over-parameterization factor k.

Model Scaling Recent works scale model dimensions like width, depth, and resolution to improve performance [22, 53]. MobileOne has similar depth scaling as MobileNet-V2, i.e. using shallower early stages where input resolution is larger as these layers are significantly slower compared to later stages which operate on smaller input resolution. We introduce 5 different width scales as seen in Table 7. Furthermore, we do not explore scaling up of input resolution as both FLOPs and memory consumption increase, which is detrimental to runtime performance on a mobile device. As our model does not have a multi-branched architecture at inference, it does not incur data movement costs as discussed in previous sections. This enables us to aggressively scale model parameters compared to competing multi-branched architectures like MobileNet-V2, EfficientNets, etc. without incurring significant latency cost. The increased parameter count enables our models to generalize well to other computer vision tasks like object detection and semantic segmentation (see Section 4). In Table 4, we compare against recent train time over-parameterization works [11–13, 19] and show that MobileOne-S1 variant outperforms RepVGG-B0 which is $\sim 3\times$ bigger.

3.4. Training

As opposed to large models, small models need less regularization to combat overfitting. It is important to have weight decay in early stages of training as demonstrated empirically by [18]. Instead of completely removing weight decay regularization as studied in [18], we find that annealing the loss incurred by weight decay regularization over the course of training is more effective. In all our experiments, we use cosine schedule [41] for learning rate. Further, we use the same schedule to anneal weight decay coefficient.

Stage	Input	# Blocks	Stride	Block Type	# Channels	MobileOne Block Parameters (α, k, act=ReLU)				
						S0	S1	S2	S3	S4
1	224×224	1	2	MobileOne-Block	$64\times\alpha$	(0.75, 4)	(1.5, 1)	(1.5, 1)	(2.0, 1)	(3.0, 1)
2	112×112	2	2	MobileOne-Block	$64\times\alpha$	(0.75, 4)	(1.5, 1)	(1.5, 1)	(2.0, 1)	(3.0, 1)
3	56×56	8	2	MobileOne-Block	$128\times\alpha$	(1.0, 4)	(1.5, 1)	(2.0, 1)	(2.5, 1)	(3.5, 1)
4	28×28	5	2	MobileOne-Block	$256\times\alpha$	(1.0, 4)	(2.0, 1)	(2.5, 1)	(3.0, 1)	(3.5, 1)
5	14×14	5	1	MobileOne-Block	$256\times\alpha$	(1.0, 4)	(2.0, 1)	(2.5, 1)	(3.0, 1)	(3.5, 1, SE-ReLU)
6	14×14	1	2	MobileOne-Block	$512\times\alpha$	(2.0, 4)	(2.5, 1)	(4.0, 1)	(4.0, 1)	(4.0, 1, SE-ReLU)
7	7×7	1	1	AvgPool	-	-	-	-	-	-
8	1×1	1	1	Linear	$512\times\alpha$	2.0	2.5	4.0	4.0	4.0

Table 7. MobileOne Network Specifications

	Baseline	+ Progressive Learning	+ Annealing Weight Decay	+ EMA
Top-1	76.4	76.8	77.3	77.4

Table 8. Ablation on various train settings for MobileOne-S2 showing Top-1 accuracy on ImageNet.

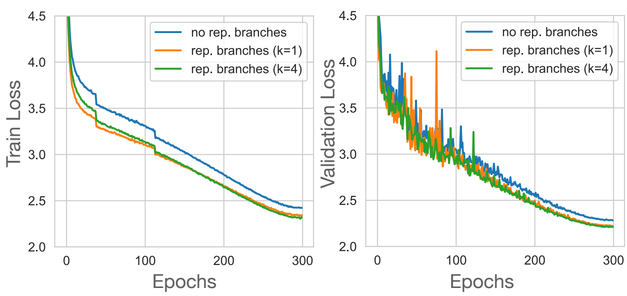


Figure 4. Plot of train and validation losses of MobileOne-S0 model. From no branches to adding re-parameterizable branches with k=1, leads to 3.4% lower train loss. Adding more branches (k=4) lowers train loss by an additional $\sim$1%. From no branches to the variant with re-parameterizable branches (k=4), validation loss improves by 3.1%

We also use the progressive learning curriculum introduced in [55]. In Table 8, we ablate over the various train settings keeping all other parameters fixed. We see that annealing the weight decay coefficient gives a 0.5% improvement.

3.5. Benchmarking

Getting accurate latency measurements on a mobile device can be difficult. On the iPhone 12, there is no command line access or functionality to reserve all of a compute fabric for just the model execution. We also do not have access to the breakdown of the round-trip-latency into categories like the network initialization, data movement, and network execution. To measure latency, we developed an iOS application using swift [35]. The application runs the models using Core ML [56]. To eliminate startup inconsistencies, the model graph is loaded, the input tensor is preallocated, and the model is run once before benchmarking begins. During benchmarking, the app runs the model many times (default is 1000) and statistic are accumulated. To achieve lowest latency and highest consistency, all other applications on the phone are closed. For the models latency seen in Table 9, we report the full round-trip latency. A large fraction of this time may be from platform processes that are not model execution, but in a real application these delays may be unavoidable. Therefore we chose to include them in the reported latency. In order to filter out interrupts from other processes, we report the minimum latency for all the models. For CPU latency, we run the models on an Ubuntu desktop with a 2.3 GHz – Intel Xeon Gold 5118 processor. For GPU latency, we compile the models using NVIDIA TensorRT library (v8.0.1.6) and run on a single RTX-2080Ti GPU with batch size set to 1. We report the median latency value out of 100 runs.

4. Experiments

Image Classification on ImageNet-1K We evaluate MobileOne models on ImageNet [10] dataset, which consists of 1.28 million training images and a validation set with 50,000 images from 1,000 classes. All models are trained from scratch using PyTorch [45] library on a machine with 8 NVIDIA GPUs. All models are trained for 300 epochs with an effective batch size of 256 using SGD with momentum [50] optimizer. We use label smoothing regularization [51] with cross entropy loss with smoothing factor set to 0.1 for all models. The initial learning rate is 0.1 and annealed using a cosine schedule [41]. Initial weight decay coefficient is set to 10^{-4} and annealed to 10^{-5} using the same cosine schedule as described in [41]. We use AutoAugment [7] to train only the bigger variants of MobileOne, i.e. S2, S3, and S4. The strength of autoaugmentation and image resolution is progressively increased during training as introduced in [55]. We list the details in supplementary material. For smaller variants of MobileOne, i.e. S0 and S1 we use standard augmentation – random resized cropping and horizontal flipping. We also use EMA (Exponential Moving Average) weight averaging with decay constant of 0.9995 for training all versions of MobileOne. At test time, all MobileOne models are evaluated on images of resolution 224×224. In Table 9, we compare against all

Model	Top-1	FLOPs (M)	Params (M)	Latency (ms) CPU	GPU	Mobile
Transformer Architectures						
Mobileformer-96 [5]	72.8	96	4.6	37.36	-	16.95
ConViT-tiny [8]	73.1	1000	5.7	28.95	-	10.99
MobileViT-S [44]	78.4	1792	5.6	30.76	-	9.21
Mobileformer-52 [5]	68.7	52	3.6	29.23	-	9.02
PiT-ti [29]	71.3	710	4.9	16.37	1.97	8.81
MobileViT-XS [44]	74.8	941	2.3	27.21	-	6.97
DeiT-tiny [57]	72.2	1300	5.9	16.68	1.78	4.78
MobileViT-XXS [44]	69.0	373	1.3	23.03	-	4.70
Convolutional Architectures						
RepVGG-B1 [13]	78.4	11800	51.8	193.7	3.17	3.73
RepVGG-A2 [13]	76.5	5100	25.5	93.43	2.41	2.41
MobileOne-S4	79.4	2978	14.8	26.60	**0.95**	**1.86**
RepVGG-B0 [13]	75.1	3100	14.3	55.97	1.45	1.82
EfficientNet-B0 [53]	77.1	390	5.3	28.71	1.35	1.72
RepVGG-A1 [13]	74.5	2400	12.8	47.15	1.42	1.68
MobileOne-S3	78.1	1896	10.1	16.47	**0.76**	**1.53**
MobileNetV2-x1.4 [46]	74.7	585	6.9	15.67	0.80	1.36
RepVGG-A0 [13]	72.4	1400	8.3	43.61	1.23	1.28
MobileNeXt-x1.4 [65]	76.1	590	6.1	18.06	1.04	1.27
MobileOne-S2	77.4	1299	7.8	14.87	**0.72**	**1.18**
MixNet-S [54]	75.8	256	4.1	40.09	2.41	1.13
MobileNetV3-L [30]	75.2	219	5.4	17.09	3.8	1.09
ShuffleNetV2-2.0 [42]	74.9	591	7.4	20.85	4.76	1.08
MNASNet-A1 [52]	75.2	312	3.9	24.06	0.95	1.00
MobileNetV2-x1.0 [46]	72.0	300	3.4	13.65	0.69	0.98
MobileNetV1 [31]	70.6	575	4.2	10.65	0.58	0.95
MobileNeXt-x1.0 [65]	74.0	311	3.4	16.04	1.02	0.92
MobileOne-S1	75.9	825	4.8	13.04	**0.66**	**0.89**
MobileNetV3-S [30]	67.4	56	2.5	10.38	3.74	0.83
ShuffleNetV2-1.0 [42]	69.4	146	2.3	16.60	4.58	0.68
MobileOne-S0	71.4	275	2.1	10.55	**0.56**	**0.79**

Table 9. Performance of various models on ImageNet-1k validation set. Note: All results are without distillation for a fair comparison. Results are grouped based on latency on mobile device. Models which could not be reliably exported either by TensorRT or Core ML Tools are annotated by "-".

recent efficient models that are evaluated on images of resolution 224×224 while having a parameter count <20 Million and trained without distillation as done in prior works like [5, 44]. FLOP counts are reported using the fvcore [17] library.

We show that even the smallest variants of transformer architectures have a latency upwards of 4ms on mobile device. Current state-of-the-art MobileFormer [5] attains top-1 accuracy of 79.3% with a latency of 70.76ms, while MobileOne-S4 attains 79.4% with a latency of only 1.86ms which is $\sim$38$\times$ faster on mobile. MobileOne-S3 has 1% better top-1 accuracy than EfficientNet-B0 and is faster by 11% on mobile. Our models have a lower latency even on CPU and GPU compared to competing methods.

Knowledge distillation Efficient models are often distilled from a bigger teacher model to further boost the performance. We demonstrate the performance of MobileOne backbones using state-of-the-art distillation recipe suggested in [47]. From Table 10, our models outperform competing models of similar or higher parameter count. Train-time overparameterization enables our models to distill to better performance even though they have similar or smaller parameter count than competing models. In fact, MobileOne-S4 outperforms even ResNet-50 model which has 72.9% more parameters. MobileOne-S0 has 0.4M less parameters at inference than MobileNetV3-Small and obtains 2.8% better top-1 accuracy on ImageNet-1k dataset.

Model	Params (M)	Latency (ms)	Top-1 Accuracy Baseline	Distillation
MobileNet V3-Small x1.0	2.5	0.83	67.4	69.7
MobileOne-S0	2.1	0.79	71.4	**72.5**
MobileNet V3-Large 1.0	5.5	1.09	75.2	76.9
MobileOne-S1	4.8	0.89	75.9	**77.4**
EfficientNet-B0	5.3	1.72	77.1	78.3
MobileOne-S2	7.8	1.18	77.4	**79.1**
ResNet-18	11.7	2.10	69.8	73.2
MobileOne-S3	10.1	1.53	78.1	**80.0**
ResNet-50	25.6	2.69	79.0	81.0
MobileOne-S4	14.8	1.86	79.4	**81.4**

Table 10. Performance of various models on ImageNet-1k validation set using MEAL-V2 [47] distillation recipe. Results of competing models are reported from [47]. Models grouped based on parameter count.

Feature backbone	mAP (↑)
MobileNetV3 [30]	22.0
MobileNetV2 [46]	22.1
MobileNetV1 [31]	22.2
MixNet [54]	22.3
MNASNet-A1 [52]	23.0
MobileVit-XS [44]	24.8
MobileViT-S [44]	27.7
MobileOne-S1	25.7
MobileOne-S2	26.6
MobileOne-S3	27.3
MobileOne-S4	29.4

(a)

Feature backbone	mIoU (↑) VOC	ADE20k
MobileNetV2-x0.5	70.2	-
MobileNetV2-x1.0	75.7	34.1
MobileViT-XXS	73.6	-
MobileViT-XS	77.1	-
MobileViT-S	79.1	-
MobileOne-S0	73.7	33.1
MobileOne-S1	77.3	35.1
MobileOne-S2	77.9	35.7
MobileOne-S3	78.8	36.2
MobileOne-S4†	80.1	38.2

(b)

Table 11. (a) Quantitative performance of object detection on MS-COCO. (b) Quantitative performance of semantic segmentation on Pascal-VOC and ADE20k datasets. †This model was trained without Squeeze-Excite layers.

Object detection on MS-COCO To demonstrate the versatility of MobileOne, we use it as the backbone feature extractor for a single shot object detector SSD [38]. Following [46], we replace standard convolutions in SSD head with separable convolutions, resulting in a version of SSD called SSDLite. The model is trained using the mmdetection library [3] on the MS COCO dataset [37]. The input resolution is set to 320×320 and the model is trained for 200 epochs as described in [44]. For more detailed hyperparameters please refer to the supplementary material. We report mAP@IoU of 0.50:0.05:0.95 on the validation set of MS COCO in Table 11. Our best model outperforms MNASNet by 27.8% and best version of MobileViT [44] by 6.1%. We show qualitative results in the supplementary material.

Semantic Segmentation on Pascal VOC and ADE 20k We use MobileOne as the backbone for a Deeplab V3 segmentation network [4] using the cvnets library [44]. The VOC models were trained on the augmented Pascal VOC dataset [16] for 50 epochs following the training procedure of [44]. The ADE 20k [64] models were trained using the same hyperparameters and augmentations. For more detailed hyperparameters, please refer to the supplementary material. We report mean intersection-over-union (mIOU) results in Table 11. For VOC, our model outperforms Mobile ViT by 1.3% and MobileNetV2 by 5.8%. Using the MobileOne-S1 backbone with a lower latency than the MobileNetV2-1.0 backbone, we still outperform it by 2.1%. For ADE 20k, our best variant outperforms MobileNetV2 by 12.0%. Using the smaller MobileOne-S1 backbone, we still outperform it by 2.9%. We show qualitative results in the supplementary material.

Robustness to corruption We evaluate MobileOne and competing models on the following benchmarks, ImageNet-A [28], a dataset that contains naturally occuring examples that are misclassified by resnets. ImageNet-R [25], a dataset that contains natural renditions of ImageNet object classes with different textures and local image statistics. ImageNet-Sketch [58], a dataset that contains black and white sketches of all ImageNet classes, obtained using google image queries. ImageNet-C [26], a dataset that consists of algorithmically generated corruptions (blur, noise) applied to the ImageNet test-set. We follow the protocol set by [43] for all the evaluations. We use pretrained weights provided by Timm Library [59] for the evaluations. From Table 12, MobileOne outperforms other efficient architectures significantly on out-of-distribution benchmarks like ImageNet-R and ImageNet-Sketch. Our model is less robust to corruption when compared to MobileNetV3-L, but outperforms MobileNetV3-L on out-of-distribution benchmarks. Our model outperforms MobileNetV3-S, MobileNetV2 variants and EfficientNet-B0 on both corruption and out-of-distribution benchmarks as seen in Table 12.

Comparison with Micro Architectures Recently [22, 36] introduced architectures that were extremely efficient in terms of FLOPS and parameter count. But architectural choices introduced in these micro architectures like [36], do not always result in lower latency models. MicroNet uses dynamic activations which are extremely inefficient as demonstrated in Table 2. In fact, smaller variants of MobileOne can easily outperform previous state-of-the-art micro architectures. Please see supplementary materials for more details on MobileOne micro architectures. In Table 13, our models have similar latency as TinyNets, but have significantly lower parameter count and better top-1 accuracy. MobileOne-μ1, is 2$\times$ smaller and has 6.3% better top-1 accuracy while having similar latency as TinyNet-E.

Model	Latency(ms)	Clean	IN-C (↓)	IN-A	IN-R	IN-SK
MobileNetV3-S	0.83	67.9	86.5	2.0	27.3	16.2
MobileOne-S0	0.79	**71.4**	**86.4**	**2.3**	**32.9**	**19.3**
MixNet-S	1.13	75.7	77.7	**3.8**	32.2	20.5
MobileNetV3-L	1.09	75.6	**77.1**	3.5	33.9	22.6
MobileNetV2-x1.0	0.98	73.0	84.1	2.1	32.5	20.8
MobileOne-S1	0.89	**75.9**	80.4	2.7	**36.7**	**22.6**
MobileNetV2-x1.4	1.36	76.5	78.9	3.7	36.0	23.7
MobileOne-S2	1.18	**77.4**	**73.6**	**4.8**	**40.0**	**26.4**
EfficientNet-B0	1.72	77.6	72.2	7.2	36.6	25.0
MobileOne-S3	1.53	78.1	71.6	7.1	**42.1**	28.5
MobileOne-S4	1.86	**79.4**	**68.1**	**10.8**	41.8	**29.2**

Table 12. Results on robustness benchmark datasets following protocol set by [43]. For ImageNet-C mean corruption error is reported (lower is better) and for other datasets Top-1 accuracy is reported (higher is better). Results are grouped following Table 9

Model	Top-1	FLOPs (M)	Params (M)	Mobile Latency (ms)
TinyNet-D [22]	67.0	52	2.3	0.51
MobileOne-μ2	**69.0**	214	1.3	**0.50**
MicroNet-M3 [36]	62.5	20	2.6	12.02
MicroNet-M2 [36]	59.4	12	2.4	9.49
TinyNet-E [22]	59.9	24	2.0	0.49
MobileOne-μ1	**66.2**	139	0.98	**0.47**
MicroNet-M1 [36]	51.4	6	1.8	3.33
MobileOne-μ0	**58.5**	68	0.57	**0.45**

Table 13. Performance of various micro-architecture models on ImageNet-1k validation set. Note, we replace swish activations with ReLU in TinyNets for a fair comparison.

5. Discussion

We have proposed an efficient, general-purpose backbone for mobile devices. Our backbone is suitable for general tasks such as image classification, object detection and semantic segmentation. We show that in the efficient regime, latency may not correlate well with other metrics like parameter count and FLOPs. Furthermore, we analyze the efficiency bottlenecks for various architectural components used in modern efficient CNNs by measuring their latency directly on a mobile device. We empirically show the improvement in optimization bottlenecks with the use of re-parameterizable structures. Our model scaling strategy with the use of re-parameterizable structures attains state-of-the-art performance while being efficient both on a mobile device and a desktop CPU.

Limitations and Future Work Although, our models are state-of-the-art within the regime of efficient architectures, the accuracy lags large models [39, 40]. Future work will aim at improving the accuracy of these lightweight models. We will also explore the use of our backbone for faster inference on other computer vision applications not explored in this work such as optical flow, depth estimation, 3D reconstruction, etc.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *Neural and Evolutionary Computing*, 2018. 4

[2] Junjie Bai, Fang Lu, Ke Zhang, et al. ONNX: Open neural network exchange. `https://github.com/onnx/onnx`, 2019. 3

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 7

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 8

[5] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 3, 7

[6] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic relu. In *16th European Conference Computer Vision (ECCV 2020)*, 2020. 3, 4

[7] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 6

[8] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. 3, 7

[9] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *arXiv preprint arXiv:2110.12894*, 2021. 1, 3

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 6

[11] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 1, 3, 4, 5

[12] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 1, 3, 4, 5

[13] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1, 3, 4, 5, 7

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 3

[15] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. 4

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 2, 8

[17] fvcore. Light-weight core library that provides the most common and essential functionality shared in various computer vision frameworks developed in fair. `https://github.com/facebookresearch/fvcore`, 2019. 7

[18] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In *Advances in Neural Information Processing Systems*, 2019. 5

[19] Shuxuan Guo, Jose M. Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. In *Advances in Neural Information Processing Systems*, 2020. 3, 4, 5

[20] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 3

[21] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. In *NeurIPS*, 2020. 3

[22] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. In *NeurIPS*, 2020. 5, 8

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 4

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[25] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. 2021. 8

[26] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 8

[27] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 4

[28] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. 2021. 8

[29] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. In *International Conference on Computer Vision (ICCV)*, 2021. 3, 7

[30] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019. 1, 2, 3, 7

[31] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017. 1, 3, 4, 7

[32] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 3, 4

[33] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[34] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size. *CoRR*, 2016. 3

[35] Apple inc. Swift programming language. https://www.swift.org, 2016. 6

[36] Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang, and Nuno Vasconcelos. Micronet: Improving image recognition with extremely low flops. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 3, 4, 8

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. 2, 7

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 7

[39] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 8

[40] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022. 8

[41] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017. 5, 6

[42] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 1, 3, 7

[43] Xiaofeng Mao, Gege Qi, Yuefeng Chen, Xiaodan Li, Ranjie Duan, Shaokai Ye, Yuan He, and Hui Xue. Towards robust vision transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 8

[44] Sachin Mehta and Mohammad Rastegari. Mobilevit: Lightweight, general-purpose, and mobile-friendly vision transformer. In *ICLR*, 2022. 1, 2, 3, 7, 8

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. 2019. 6

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 1, 3, 7

[47] Zhiqiang Shen and Marios Savvides. Meal v2: Boosting vanilla resnet-50 to 80%+ top-1 accuracy on imagenet without tricks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 7

[48] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[49] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1

[50] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, 2013. 6

[51] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6

[52] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3, 7

[53] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning (PMLR)*, 2019. 2, 3, 5, 7

[54] Mingxing Tan and Quoc V. Le. Mixconv: Mixed depthwise convolutional kernels. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, 2019. 2, 3, 7

[55] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. 3, 6

[56] Core ML Tools. Use Core ML Tools to convert models from third-party libraries to Core ML. https://coremltools.readme.io/docs, 2017. 1, 3, 6

[57] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 3, 7

[58] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, 2019. 8

[59] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 8

[60] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021. 3

[61] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross B. Girshick. Early convolutions help transformers see better. *CoRR*, abs/2106.14881, 2021. 3

[62] Jerrold H Zar. Spearman rank correlation. *Encyclopedia of biostatistics*, 7, 2005. 3

[63] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018. 3

[64] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2, 8

[65] Daquan Zhou, Qibin Hou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. Rethinking bottleneck structure for efficient mobile network design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 3, 7