# Supplementary Material – MobileOne: An Improved One millisecond Mobile Backbone

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

Apple

## A. Figures

Figure 1 from the main paper has been enlarged in Figures 1, 2, 3.

## B. Benchmarking

We treat MobileNetV3 [5] in a special way since their H-swish operator is optimized for certain hardware platforms and not for others. Howard et al. [5] show that H-swish can obtain similar performance as ReLU when platform specific optimizations are applied. Therefore, while benchmarking for latency, we replace the H-swish layers with ReLU layers and then report the latency of MobileNetV3.

### B.1. Additional Benchmarks

We have shown the efficiency of our model with comparisons on CPU, desktop GPU (RTX-2080Ti) and Mobile (iPhone 12). Additionally, in Table 1, we port existing architectures to Pixel-6 TPU and compare with our model. We observe that MobileOne achieves state-of-the-art accuracy-latency trade-off on TPU as well.

| | | **Latency (ms) ↓** | | | |
|---|---|---|---|---|---|
| **Model** | Top1 ↑ | CPU (x86) | iPhone12 (ANE) | TensorRT (2080Ti) | Pixel-6† (TPU) |
| RepVGG-B2 | 78.8 | 492.8 | 6.38 | 4.79 | 6.83 |
| RepVGG-B1 | 78.4 | 193.7 | 3.73 | 3.17 | 4.28 |
| RepVGG-A2 | 76.5 | 93.43 | 2.41 | 2.41 | 2.28 |
| **MobileOne-S4** | 79.4 | 26.6 | 1.86 | 0.95 | 2.17 |
| EfficientNet-B0 | 77.1 | 28.71 | 1.72 | 1.35 | 2.49 |
| **MobileOne-S3** | 78.1 | 16.47 | 1.53 | 0.76 | 1.28 |
| RepVGG-B0 | 75.1 | 55.97 | 1.82 | 1.42 | 1.43 |
| RepVGG-A1 | 74.5 | 47.15 | 1.68 | 1.42 | 1.21 |
| **MobileOne-S2** | 77.4 | 14.87 | 1.18 | 0.72 | 1.07 |
| RepVGG-A0 | 72.4 | 43.61 | 1.23 | 1.28 | 1.01 |
| MobileNetV3-L | 75.2 | 17.09 | 1.09 | 3.8 | 1.01 |
| MobileNetV2-x1.4 | 74.7 | 15.67 | 1.36 | 0.8 | 0.98 |
| MNASNet-A1 | 75.8 | 24.06 | 1.00 | 0.95 | 0.88 |
| MobileNetV2-x1.0 | 72.0 | 13.65 | 0.98 | 0.69 | 0.77 |
| **MobileOne-S1** | 75.9 | 13.04 | 0.89 | 0.66 | 0.79 |
| MobileNetV3-S | 67.4 | 10.38 | 0.83 | 3.74 | 0.67 |
| ShuffleNetV2-x1.0 | 69.4 | 16.6 | 0.68 | 4.58 | - |
| MobileNetV1 | 70.6 | 10.65 | 0.95 | 0.58 | 0.73 |
| **MobileOne-S0** | 71.4 | 10.55 | 0.79 | 0.56 | 0.59 |

Table 1. Comparison with mobile architectures on Intel Xeon CPU, NVIDIA 2080Ti GPU, iPhone 12 and Pixel-6. "†" denotes models on Pixel-6 TPU, where weights and activations were converted to `int8` format. For all other compute platforms, models were evaluated in `fp16` format.

## C. Image Classification

### C.1. Training details

All models are trained from scratch using PyTorch [10] library on a machine with 8 NVIDIA A100 GPUs. All models are trained for 300 epochs with an effective batch size of 256 using SGD with momentum [11] optimizer. We follow progressive training curriculum [13] for faster training and better generalization. Throughout training the image resolution and the augmentation strength($\alpha$) is gradually increased, see Table 2. The magnitude for augmentations in AutoAugment [3] policy are between 0-9, we simply multiply $\alpha$ with this value to simulate variable strength of autoaugmentation. AutoAugment [3] is used to train only the bigger variants of MobileOne, i.e. S2, S3, and S4. For smaller variants of MobileOne, i.e. S0 and S1 we use standard augmentation – random resized cropping and horizontal flipping. We use label smoothing regularization [12] 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 [8]. Initial weight decay coefficient is set to $10^{-4}$ and annealed to $10^{-5}$ using the same cosine schedule. We also use EMA (Exponential Moving Average)

| **Epoch Range** | **Image Resolution** | **AutoAugment Strength** |
|---|---|---|
| 0 - 38 | 160 | 0.3 |
| 39 - 113 | 192 | 0.6 |
| 114 - 300 | 224 | 1.0 |

Table 2. Progressive training settings. AutoAugment is used only for training MobileOne-S2,S3,S4 variants.

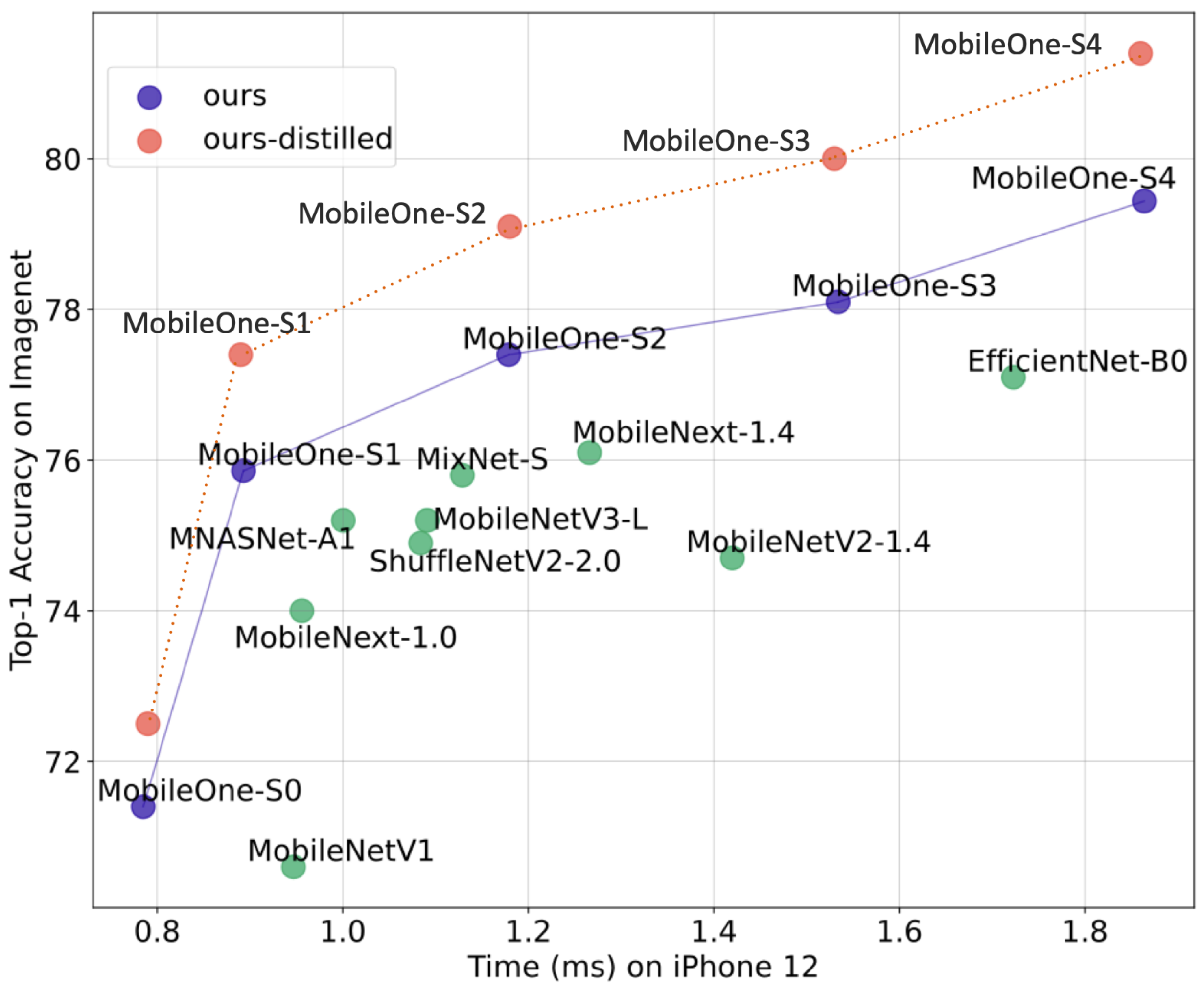

Figure 1. Top 1 accuracy vs Latency on iPhone 12. Corresponds to Figure 1a in the main paper.

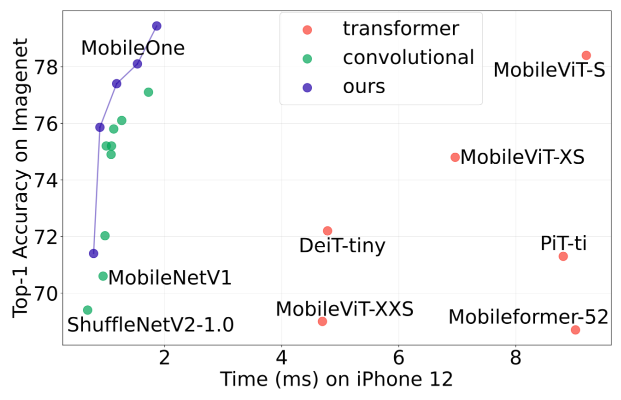

Figure 2. Zoomed out (a). Corresponds to Figure 1b in the main paper.

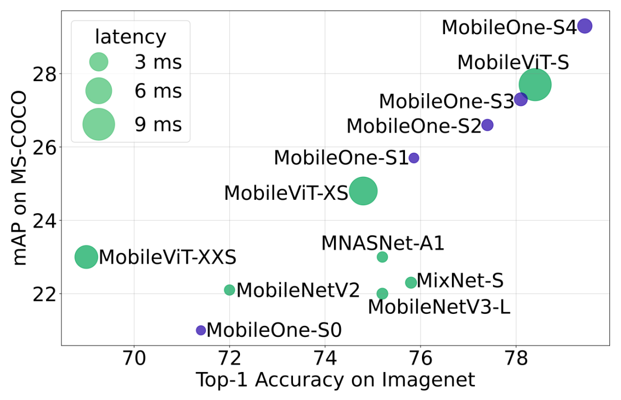

Figure 3. Top-1 accuracy vs mAP. Corresponds to Figure 1c in the main paper.

| Model | Top-1 Accuracy | Mobile Latency(ms) | Training Recipe |
|---|---|---|---|
| MobileOne-S4 (Ours) | 79.4 | 1.86 | CosLR + EMA + AA + PL + AWD |
| MobileOne-S3 (Ours) | 78.1 | 1.53 | CosLR + EMA + AA + PL + AWD |
| EfficientNet-B0 | 77.1 | 1.72 | Baseline reported by respective authors |
| EfficientNet-B0 | 77.4 | 1.72 | WCosLR + EMA + RA + RandE + DropPath + Dropout (Baseline reproduced) |
| EfficientNet-B0 | 77.8 | 1.72 | WCosLR + EMA + RA + RandE + DropPath + Dropout + PL + AWD |
| EfficientNet-B0 | 74.9 | 1.72 | CosLR + EMA + AA + PL + AWD |
| MobileOne-S2 (Ours) | 77.4 | 1.18 | CosLR + EMA + AA + PL + AWD |
| MobileNetV2 $\times$1.4 | 74.7 | 1.36 | Baseline reported by respective authors |
| MobileNetV2 $\times$1.4 | 75.7 | 1.36 | WCosLR + EMA + RA + RandE + DropPath + Dropout (Baseline reproduced) |
| MobileNetV2 $\times$1.4 | 76.2 | 1.36 | WCosLR + EMA + RA + RandE + DropPath + Dropout + PL + AWD |
| MobileNetV2 $\times$1.4 | 76.0 | 1.36 | CosLR + EMA + AA + PL + AWD |
| MobileOne-S1 (Ours) | 75.9 | 0.89 | CosLR + EMA + PL + AWD |
| MixNet-S | 75.8 | 1.13 | Baseline reported by respective authors |
| MixNet-S | 75.6 | 1.13 | WCosLR + EMA + DropPath (Baseline reproduced) |
| MixNet-S | 75.4 | 1.13 | WCosLR + EMA + DropPath + PL + AWD |
| MixNet-S | 75.5 | 1.13 | CosLR + EMA + PL + AWD |
| MobileNetV3-L | 75.2 | 1.09 | Baseline reported by respective authors |
| MobileNetV3-L | 75.4 | 1.09 | WCosLR + EMA + RA + RandE + DropPath + Dropout + LR Noise (Baseline reproduced) |
| MobileNetV3-L | 75.6 | 1.09 | WCosLR + EMA + RA + RandE + DropPath + Dropout + LR Noise + PL + AWD |
| MobileNetV3-L | 72.5 | 1.09 | CosLR + EMA + AA + PL + AWD |
| MobileNetV2 $\times$1.0 | 72.0 | 0.98 | Baseline reported by respective authors |
| MobileNetV2 $\times$1.0 | 72.9 | 0.98 | WCosLR + EMA (Baseline reproduced) |
| MobileNetV2 $\times$1.0 | 73.0 | 0.98 | WCosLR + EMA + PL + AWD |
| MobileNetV1 | 70.6 | 0.95 | Baseline reported by respective authors |
| MobileNetV1 | 72.7 | 0.95 | CosLR + EMA (Baseline reproduced) |
| MobileNetV1 | 73.7 | 0.95 | CosLR + EMA + PL + AWD |

| Legend | |
|---|---|
| AA | AutoAugment |
| RA | RandAugment |
| PL | Progressive Learning |
| AWD | Annealing Weight Decay |
| RandE | Random Erasing |
| EMA | Exponential Moving Average |
| CosLR | Cosine learning rate schedule |
| WCosLR | Cosine learning rate schedule with Warmup |
| LR Noise | Learning Rate Noise schedule in Timm |

Table 3. Top-1 Accuracy on ImageNet-1k for various training recipes.

weight averaging with decay constant of 0.9995 for training all versions of MobileOne.

## C.2. Analysis of Training Recipes

Recent models introduce their own training recipe including regularization techniques to train them to competitive accuracies. We ablate over some of the commonly used recipes to train EfficientNet, MobileNetV3-L, MixNet-S, MobileNetV2 and MobileNetV1 in Table 3. Mainly, we report the following,

- Results from original training recipes of the respective models. (baselines)
- Results from training the models using recipe used to train MobileOne models.
- Results obtained by adding EMA, Progressive Learning (PL) and Annealing Weight decay (AWD) to the original recipe proposed by respective works.

All runs below have been reproduced using Timm library [14]. For a fair comparison all models are trained for 300 epochs. From Table 3, we observe that our models use less regularization techniques as opposed to competing models like EfficientNet, MobileNetV3-L and MixNet-S to reach competitive accuracies. When we apply our training recipe to the competing models, there is no improvement in models like EfficientNet, MobileNetV3-L and MixNet-S. There are slight improvements in MobileNetV2 and MobileNetV1. However, the accuracy at iso-latency gap between our models is still large. When progressive

| **Stage** | **Input** | **Stride** | **Block Type** | **# Channels** | **(# Blocks, $\alpha$, $k$) act=ReLU** | | |
|---|---|---|---|---|---|---|---|
| | | | | | **$\mu$0** | **$\mu$1** | **$\mu$2** |
| 1 | 224 × 224 | 2 | MobileOne-Block | 64×$\alpha$ | (1, 0.75, 3) | (1, 0.75, 2) | (1, 0.75, 2) |
| 2 | 112 × 112 | 2 | MobileOne-Block | 64×$\alpha$ | (2, 0.75, 3) | (2, 0.75, 2) | (2, 0.75, 2) |
| 3 | 56 × 56 | 2 | MobileOne-Block | 128×$\alpha$ | (4, 0.5, 3) | (6, 0.75, 2) | (6, 1.0, 2) |
| 4 | 28 × 28 | 2 | MobileOne-Block | 256×$\alpha$ | (3, 0.5, 3) | (4, 0.75, 2) | (4, 1.0, 2) |
| 5 | 14 × 14 | 1 | MobileOne-Block | 256×$\alpha$ | (3, 0.5, 3) | (4, 0.75, 2) | (4, 1.0, 2) |
| 6 | 14 × 14 | 2 | MobileOne-Block | 512×$\alpha$ | (1, 0.75, 3) | (1, 1.0, 2) | (1, 1.0, 2) |
| 7 | 7 × 7 | 1 | AvgPool | - | - | - | - |
| 8 | 1 × 1 | 1 | Linear | 512×$\alpha$ | 0.75 | 1.0 | 1.0 |

Table 4. MobileOne micro variant specifications.

learning and annealing weight decay is used with baseline recipes, we obtain additional improvements, for example MobileNetV1, gets 1% improvement and MobileNetV2 ×1.4 gets 0.5% improvement.

## C.3. Sensitivity to Random Seeds

Our model and training runs are stable and give similar performance with different random seeds, see Table 5.

| **Model** | **Run #1** | **Run #2** |
|---|---|---|
| MobileOne-S0 | 71.402 | 71.304 |
| MobileOne-S1 | 75.858 | 75.877 |
| MobileOne-S2 | 77.372 | 77.234 |
| MobileOne-S3 | 78.082 | 78.008 |
| MobileOne-S4 | 79.436 | 79.376 |

Table 5. Runs from 2 different seeds for all variants of MobileOne

## D. Micro Architectures

In Table 4, we provide specifications for micro variants of MobileOne introduced in Table 13 of main paper. Rather than optimizing for FLOPs, as done in [4,6] we sample variants that are significantly smaller in parameter count and use trivial overparameterization to train these architectures to competitive accuracies.

### D.1. Effectiveness of Overparameterization

We find that additional overparameterization branches benefits smaller variants more than it does for larger variants. In our experiments, we found that smaller variants improve consistently with additional overparameterization branches. Note, for all the experiments in Table 6, we use the same hyperparameters as described in Section 4 of main paper.

| | $k = 1$ | $k = 2$ | $k = 3$ |
|---|---|---|---|
| MobileOne-$\mu$1 | 65.7 | 66.2 | 65.9 |
| MobileOne-$\mu$2 | 68.6 | 69.0 | 68.8 |
| MobileOne-S0 | 70.9 | 70.7 | 71.3 |

Table 6. Effect of over-parametrization factor $k$ on MobileOne variants. Top-1 accuracy on ImageNet is reported.

## E. Object Detection

### E.1. Training details

SSDLite models were trained for 200 epochs using cosine learning rate schedule with warmup, following [9]. Linear warmup schedule with a warmup ratio of 0.001 for 4500 iterations was used. Image size of 320×320 was used for both training and evaluation, following [9]. We used SGD with momentum optimizer [11] with an initial learning rate of 0.05, momentum of 0.9 and weight decay of 0.0001 for all the models. We use an effective batchsize of 192, following [1]. The models were trained on a machine with 8 NVIDIA A100 GPUs.

### E.2. Qualitative Results

Visualizations in Figure 4 are generated using `image_demo.py` [1] with default thresholds in MMDetection library [1]. We compare MobileNetV2-SSDLite with MobileOne-S2-SSDLite which have similar latencies. Our model outperforms MobileNetV2-SSDLite in detecting small and large objects. In the first row, our model detects the potted plants amongst all the clutter in the scene. In the second row, our model detects both the dog and frisbee as opposed to MobileNetV2. In the third row, our model detects the tennis racket and the ball even though they are blurry. In the remaining rows, our model consistently detects both small and large foreground objects as opposed to MobileNetV2.

## F. Semantic Segmentation

### F.1. Training details

We use the MobileViT repository [9] to train our semantic segmentation models and adopt their hyperparameter settings. Both VOC and ADE20k segmentation models were trained for 50 epochs using cosine learning rate with a

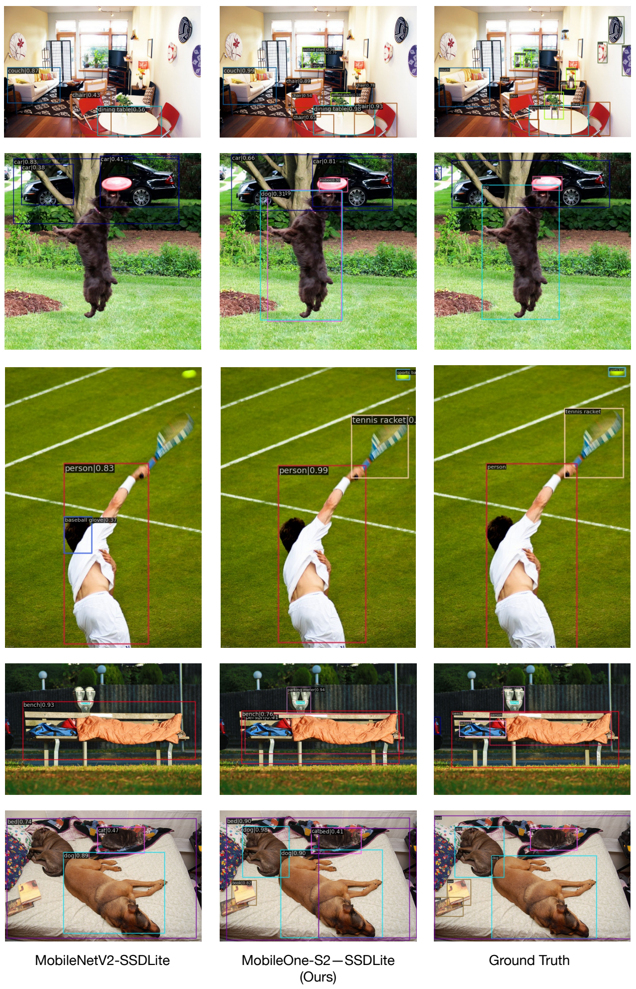

Figure 4. Qualitative comparison of MobileOne-S2-SSDLite (middle) against MobileNetV2-SSDLite (left) and ground truth (right). The two models have similar latency.

maximum learning rate of $10^{-4}$ and minimum learning rate of $10^{-6}$. We use 500 warmup iterations. The segmentation head has a learning rate multiplier of 10. EMA is used with a momentum of $5 \times 10^{-4}$. We use AdamW optimizer [7] with weight decay of 0.01. For VOC, the model is trained on both MS-COCO and VOC data simultaneously following Mehta et al [9]. For both VOC and ADE20k, the only augmentations used are random resize, random crop, and horizontal flipping.

## F.2. Qualitative Results

We provide qualitative results for semantic segmentation in Figure 5. Our method performs better than MobileViT-S-DeepLabV3 as shown. In row 1, we show that MobileViT-S misclassifies background as airplane. In row 2 and row 6, our method is able to resolve fine details such as the leg of the horse and tiny birds. In row 3, MobileViT-S misclassfies the couch. In row 4, our method is able to segment large foreground object at a close-up view. In row 5, our method segments small objects such as the buses.

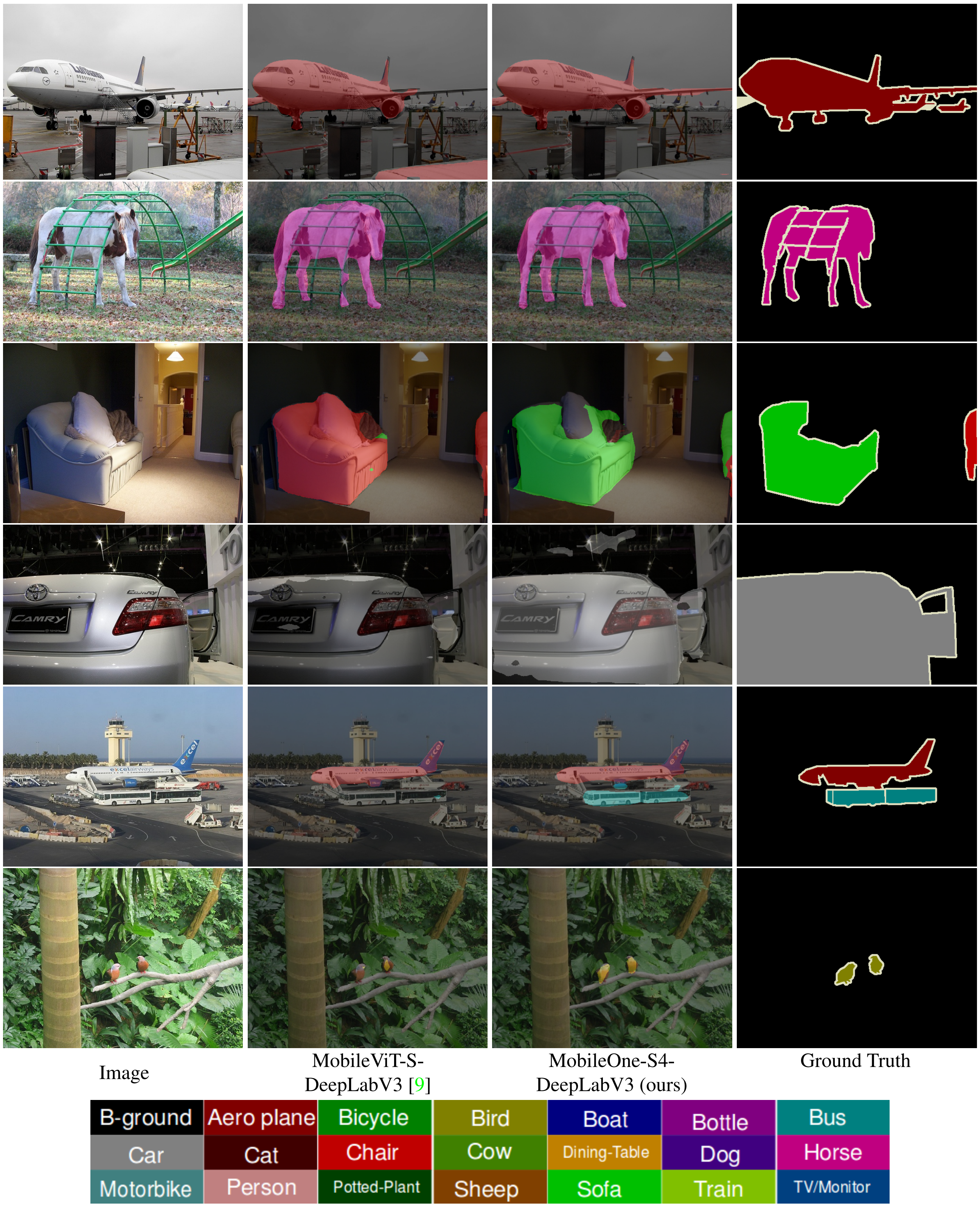

Figure 5. Qualitative results on semantic segmentation. Legend reproduced from DeepLab [2].

## References

[1] 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. 4

[2] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 6

[3] 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. 1

[4] 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. 4

[5] 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

[6] 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. 4

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 5

[8] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017. 1

[9] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *ICLR*, 2022. 4, 5, 6

[10] 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. 1

[11] 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. 1, 4

[12] 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. 1

[13] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. 1

[14] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 3