FAST NEURAL NETWORK ADAPTATION VIA PARAMETERS REMAPPING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks achieve remarkable performance in many computer vision tasks. However, for most semantic segmentation (seg) and object detection (det) tasks, the backbone of the network directly reuses the network manually designed for classification tasks. Utilizing a network pre-trained on ImageNet as the backbone has been a popular practice for seg/det challenges. However, because of the gap between different tasks, adapting the network directly to the target task could bring performance promotion. Some recent neural architecture search (NAS) methods search for the backbone of seg/det networks. ImageNet pre-training of the search space representation or the searched network bears huge computational cost. In this paper, we propose a fast neural network adaptation method FNA, which can adapt the manually designed network on ImageNet to the new seg/det tasks efficiently. We adopt differentiable NAS to adapt the architecture of the network. We first expand the manually designed network to a super network which is the representation of the search space. Then we successively conduct the adaptation on the architecture-level and parameter-level. Our designed parameters-remapping paradigm accelerates the adaptation process. Our experiments include both seg and det tasks. We conduct adaptation on the MobileNetV2 network. FNA demonstrates performance promotion compared with both manually and NAS designed networks. The total computational cost of FNA is much less than many *SOTA* seg/det NAS methods, 1737$\times$ less than DPC, 6.8$\times$ less than Auto-DeepLab and 7.4$\times$ less than DetNAS.

1 INTRODUCTION

Deep neural networks (DNNs) have achieved great success in recent years. Especially, they are widely applied in computer vision tasks, e.g., image classification (Krizhevsky et al., 2012; He et al., 2016; Howard et al., 2017), segmentation (Long et al., 2015; Ronneberger et al., 2015; Chen et al., 2017b) and object detection (Ren et al., 2015; Liu et al., 2016; Lin et al., 2017; He et al., 2017), etc. Image classification always serves as a fundamental task for neural network designing. In general, one network elaborately designed and pre-trained on the classification task is directly used as the backbone of the segmentation or detection network. The fine-tuning technique is usually utilized to adapt the parameters of the backbone network to the new task. There is no doubt that the backbone plays an important role in segmentation or detection tasks. However, there is a gap between different tasks which leads to the different principles of architecture designing. For example, segmentation tasks focus on denser features of images and object detection tasks need to take both localization and classification information for each instance. Therefore, neural architectures designed for classification tasks do not suffice so well for segmentation or detection tasks. To tackle this problem, some works (Li et al., 2018; Wang et al., 2019) propose to improve the backbone architecture designing for better performance on the target tasks.

Handcrafted neural architecture design is inefficient and may not find the best networks. Recently, neural architecture search (NAS) methods (Zoph et al., 2017; Pham et al., 2018; Liu et al., 2018; 2019b; Cai et al., 2019) aim at designing the architectures of DNNs automatically, which can alleviate the burden of human experts. Some works (Liu et al., 2019a; Zhang et al., 2019; Chen et al., 2019b) propose to use NAS methods to specialize in architecture designing for the backbones of segmentation or detection networks. However, pre-training has always been a non-negligible but costly procedure. Though it is demonstrated that pre-training is not so necessary recently (He et al.,

2018), training from scratch on the target task still takes more iterations than fine-tuning from the pre-trained model. In architecture search methods, pre-training becomes more intractable because there are millions of architectures in the search space. It is impractical to pre-train every backbone of the architectures. One-shot search methods (Brock et al., 2017; Bender et al., 2018; Chen et al., 2019b) integrate all sub-architectures in one super network while pre-training the super network and the searched network both bears large computational cost.

As ImageNet (Deng et al., 2009) pre-training has been an incredibly common manner in many computer vision tasks, there are lots of models trained on ImageNet available. To take full advantage of these models, we propose a fast neural network adaptation method (FNA) based on a parameters-remapping paradigm. Our method can adapt one network to a new task with low computational cost. Fig. 1 shows the whole framework. The adaptation acts both on the architecture- and parameters-level. We adopt the differentiable NAS method (Liu et al., 2019b; Wu et al., 2018; Cai et al., 2019) to implement the architecture-level adaptation. We select a manually designed network (MobileNetV2 (Sandler et al., 2018) in our experiments) as the `seed network`, which is pre-trained on ImageNet. With the parameters-remapping paradigm, the seed network can be expanded into a super network which is the continuous representation of the search space. The new parameters in the super network can be initialized as one mapping of that in the seed network. Benefitting from the parameters-remapping paradigm, the differentiable architecture search process can be fast carried out directly on the target task. With the NAS based architecture adaptation, we obtain a new architecture targeted at the new task. Then we conduct adaptation on the parameter-level. Similarly, we remap the parameters of the seed network on the target network. Then the target network can be directly fine-tuned on the target task with no need for pre-training once more.

Our proposed FNA method aims at adapting the neural network to new tasks fast and efficiently. Based on the parameters-remapping mechanism, FNA takes full advantage of the pre-trained parameters of the seed network. The remapping acts on both the NAS process and parameters fine-tuning, which greatly degrades the computational cost of adaptation. The effectiveness and efficiency of our method are demonstrated by experiments on both segmentation and detection tasks. We adapt the manually designed network MobileNetV2 (Sandler et al., 2018) to segmentation framework DeepLabv3 (Chen et al., 2017b), detection framework RetinaNet (Lin et al., 2017) and SSDLite (Liu et al., 2016; Sandler et al., 2018). Networks adapted by FNA surpass both manually designed and NAS searched networks in terms of both performance and model FLOPs. Compared to NAS methods, FNA takes much less computational cost, 1737$\times$ less than DPC, 6.8$\times$ less than Auto-DeepLab and 7.4$\times$ less than DetNAS.

2 Related Work

Neural Architecture Search With reinforcement learning (RL) and evolutionary algorithm (EA) being applied to NAS methods, many works (Zoph & Le, 2016; Zoph et al., 2017; Pham et al., 2018; Tan et al., 2018; Real et al., 2018) make great progress in promoting the performances of neural networks. Recently, NAS methods based on one-shot model (Brock et al., 2017; Bender et al., 2018) or differentiable representations (Liu et al., 2019b; Cai et al., 2019; Wu et al., 2018) achieve remarkable results with low search cost. We use the differentiable NAS method to implement architecture adaptation, which adjusts the architecture of the backbone network automatically.

Backbone Design As deep neural network (Simonyan & Zisserman, 2014; Szegedy et al., 2016; He et al., 2016) designing develops, the backbone of segmentation or detection evolves accordingly. Some works improve the backbone architectures by modifying existing networks. PeleeNet (Wang et al., 2018) proposes a variant of DenseNet (Huang et al., 2017) for more efficient detection system on mobile devices. DetNet (Li et al., 2018) apply dilated convolution (Yu & Koltun, 2016) on the backbone to enlarge the receptive field. BiSeNet (Yu et al., 2018) and HRNet (Wang et al., 2019) design multiple paths to learn both high- and low- resolution representations. Some works (Liu et al., 2019a; Zhang et al., 2019; Chen et al., 2019b) employ NAS methods to design better backbone architectures.

Parameters Remapping Net2Net (Chen et al., 2015) proposes the function-preserving transformations to remap the parameters of one network to a new deeper or wider network. This remapping mechanism accelerates the training of the new larger network and achieves great performances. Following this manner, EAS (Cai et al., 2018) extends the parameters-remapping concept to neural architecture search. Moreover, some NAS works (Pham et al., 2018; Fang et al., 2019; Elsken et al.,

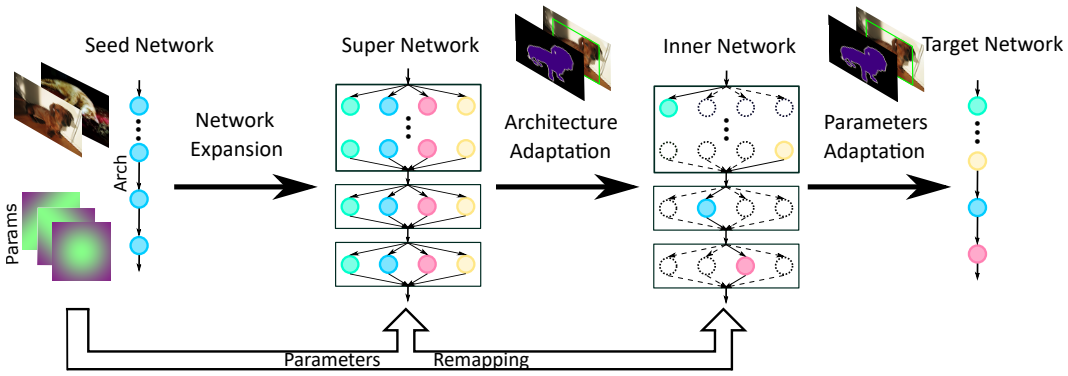


Figure 1: The framework of our proposed FNA. We select an artificially designed network as the seed network $\mathcal{N}_s$. Then we expand $\mathcal{N}_s$ to a super network $\mathcal{N}_{sup}$ which is a continuously relaxed representation of the search space. We utilize differentiable NAS method to start the architecture adaptation with the super network and then obtain an inner network $\mathcal{N}_i$ with the target architecture. Finally, we adapt the parameters of $\mathcal{N}_i$ to get the target network $\mathcal{N}_t$. The parameters of $\mathcal{N}_s$ are remapped to both $\mathcal{N}_{sup}$ and $\mathcal{N}_i$.

2019) apply parameters sharing on child models to accelerate the search process. Our parameters-remapping paradigm makes the network adaptation get rid of the cost of ImageNet pre-training and largely decreases the computational cost.

3 METHOD

The artificially-designed network MobileNetV2 (Sandler et al., 2018) has been widely used in many computer vision tasks because of its high performance and low FLOPs. In this paper, we select the MobileNetV2 (Sandler et al., 2018) network as the seed network for its generality. To adapt the network to segmentation and detection tasks, we adjust the architecture elements on three levels, i.e. convolution kernel size, depth and width of the network. In this section, we first describe the parameters-remapping paradigm. Then we explain the whole procedure of the network adaptation.

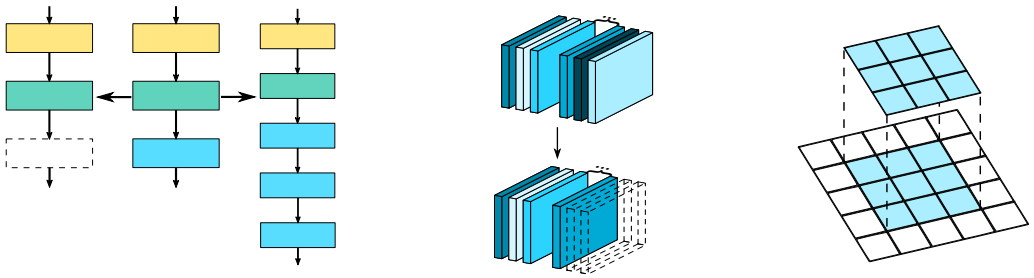
3.1 PARAMETERS REMAPPING

We define parameters-remapping as one paradigm which aims at mapping the parameters of one seed network to another one. We denote the seed network as $\mathcal{N}_s$ and the new network as $\mathcal{N}_n$, whose parameters are denoted as $\boldsymbol{W}_s$ and $\boldsymbol{W}_n$ respectively. The remapping paradigm is illustrated in the following three aspects. The remapping on the depth-level is firstly carried out and the remapping on the kernel- and width- level is conducted simultaneously. Moreover, we study different remapping manners in the experiments.

Remapping on Depth-level We extend the depth settings in our adaptation process. In other word, we adjust the number of inverted residual blocks (Sandler et al., 2018) (MBConv) in every stage of the network. We assume that one stage in the seed network $\mathcal{N}_s$ has l layers. Its parameters of each layer can be denoted as $\{\boldsymbol{W}_s^{(1)}, \boldsymbol{W}_s^{(2)}, \ldots, \boldsymbol{W}_s^{(l)}\}$. Similarly, we assume that the corresponding stage with m layers in the new network $\mathcal{N}_n$ has parameters $\{\boldsymbol{W}_n^{(1)}, \boldsymbol{W}_n^{(2)}, \ldots, \boldsymbol{W}_n^{(m)}\}$. The remapping process on the depth-level is shown in Fig. 2(a). The parameters of layers in $\mathcal{N}_n$ which also exit in $\mathcal{N}_s$ are just copied from $\mathcal{N}_s$. The parameters of new layers are all copied from the last layer in the stage of $\mathcal{N}_s$. It is formulated as

$$\begin{aligned} f(i) &= max(i, l), \\ W_n^{(i)} &= W_s^{(f(i))}, \quad \forall 1 \leq i \leq m. \end{aligned} \tag{1}$$

Remapping on Width-level In the MBConv block of MobileNetV2 (Sandler et al., 2018) network, the first point-wise convolution expands the low-dimensional features to a high dimension. This practice can be utilized for expanding the width and capacity of one neural network. We adapt



(a) Depth level (b) Width level (c) Kernel level

Figure 2: Our parameters are remapped on three levels. (a) shows the depth-level remapping. The parameters of existing corresponding layers are copied from the original network. The parameters of new layers are copied from the last layer in the original network. (b) shows the width-level remapping. For each channels-diminished dimension, the parameters are copied from the original existing ones. (c) shows the kernel-level remapping. The original parameters are mapped to the central part of the new larger kernel. The values of the other parameters are assigned with 0.

the width of the network by setting smaller expansion ratios. We denote the parameters of the convolution in $\mathcal{N}_s$ as $\boldsymbol{W}^s \in \mathbb{R}^{p\times q\times h\times w}$ and that in $\mathcal{N}_n$ as $\boldsymbol{W}^n \in \mathbb{R}^{r\times s\times h\times w}$, where $r \leq p$ and $s \leq q$. As is shown in Fig. 2(b), on the width-level, we just map the first r/s channels of parameters in $\mathcal{N}_s$ to the narrower one in $\mathcal{N}_n$. This remapping strategy is utilized for parameters sharing in EAT-NAS (Fang et al., 2019) as well. It can be formulated as

$$\boldsymbol{W}^n_{i,j,:,:} = \boldsymbol{W}^s_{i,j,:,:}, \quad \forall 1 \leq i \leq r, 1 \leq j \leq s. \tag{2}$$

Remapping on Kernel-level The kernel size is commonly set as 3×3 in most artificially-designed networks. To expand the receptive field and capture more abundant features in segmentation and detection tasks, we introduce larger kernel size settings in the adaptation process. As Fig. 2(c) shows, to expand the 3×3 kernel to a larger one, we assign the parameters of the central 3×3 region in the large kernel with the values of the original 3×3 kernel. The values of the other region surrounding the central part are assigned with 0. We denote the parameters of the original 3×3 kernel as $\boldsymbol{W}^{3\times3}$ and the larger $k \times k$ kernel as $\boldsymbol{W}^{k\times k}$. The remapping process on kernel-level can be formulated as follows,

$$\boldsymbol{W}^{k\times k}_{:,:,h,w} = \begin{cases} \boldsymbol{W}^{3\times3}_{:,:,h,w} & \text{if}(k-3)/2 < h,w \leq (k+3)/2 \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where h, w denote the indices of the spatial dimension. This design principle conforms to the function-preserving concept (Chen et al., 2015), which accelerates and stabilizes the optimization of the new network.

3.2 Neural Network Adaptation

We divide our neural network adaptation into three steps. Fig. 1 demonstrates the whole adaptation procedure. We select MobileNetV2 as the seed network $\mathcal{N}_s$. Firstly, we expand the seed network to a super network which is the representation of the search space in the latter architecture adaptation process. Secondly, we utilize the differentiable NAS method to achieve network adaptation on the architecture-level and obtain an inner network $\mathcal{N}_i$ with the target architecture. Finally, we adapt the parameters of the inner network and obtain the target network $\mathcal{N}_t$. The aforementioned parameters-remapping method is applied to both network expansion and parameters adaptation.

Network Expansion We expand the seed network to a super network by introducing more options for architecture elements. For every MBConv layer, we allow for more kernel size settings $\{3, 5, 7\}$ and more expansion ratios $\{3, 6\}$. As most differentiable NAS methods (Liu et al., 2019b; Cai et al., 2019; Wu et al., 2018) do, we set every layer as a weighted sum of all candidate operations.

$$\overline{o}^{(i)}(x) = \sum_{o\in O} \frac{exp(\alpha_o^{(i)})}{\sum_{o'\in O} exp(\alpha_{o'}^{(i)})} o(x), \tag{4}$$

where O denotes the operation set, α_o^i denotes the architecture parameter of operation o in the ith layer, and x denotes the input tensor. We extend more layers in one stage in the network and add the identity connection to the operation set for depth adjustment. After expanding the architecture of the network, we remap the parameters of the seed network to the super network based on the paradigm illustrated in Sec. 3.1. This remapping strategy prevents the huge cost of ImageNet pre-training of the all the sub-networks or the super network.

Architecture Adaptation We start the differentiable NAS process with the expanded super network directly on the target task, e.g., semantic segmentation or object detection. We first fine-tune operation weights of the super network for some epochs on the training dataset. After the weights are sufficiently trained, we start alternating the optimization of operation weights w with $\frac{\partial \mathcal{L}}{\partial w}$ and architecture parameters α with $\frac{\partial \mathcal{L}}{\partial \alpha}$. To accelerate the search process and decouple the parameters of different sub-networks, we only sample one path in each iteration of operation weight optimization. We sample the path according to the distribution of architecture parameters. With the optimization finished, we use the architecture parameters α to derive the target architecture.

Parameters Adaptation After architecture adaptation, we obtain an inner network $\mathcal{N}_i$ with the architecture targeted at the new task. To accommodate the new segmentation or detection tasks, the target architecture becomes quite different from that of the seed network $\mathcal{N}_s$ (which is primitively designed for the image classification task). Unlike conventional training strategy, we discard the cumbersome pre-training process of $\mathcal{N}_i$ on ImageNet. We remap the parameters of $\mathcal{N}_s$ to $\mathcal{N}_i$ utilizing the method described in Sec. 3.1. Then we directly fine-tune $\mathcal{N}_i$ on the target task and obtain the final target network $\mathcal{N}_t$.

4 EXPERIMENTS

We select the ImageNet pre-trained MobileNetV2 model as the seed network and apply our FNA method on both semantic segmentation and object detection tasks. In this section, firstly we give the implementation details of our experiments; then we report and analyze the network adaptation results; finally, we perform ablation studies to validate the effectiveness of the parameters-remapping paradigm and compare different parameters remapping implementations.

4.1 ADAPTATION ON SEMANTIC SEGMENTATION

The semantic segmentation experiments are conducted on the Cityscapes (Cordts et al., 2016) dataset, which contains 2975 images for the training and 500 for the validation dataset. The image size is 1024×2048. In the architecture adaptation process, we map the seed network to the super network, which is used as the backbone of DeepLabv3 (Chen et al., 2017b). The output feature map is down-sampled by the factor of 16. Depthwise-separable convolution is used in the ASPP module (Chen et al., 2017a;b). We randomly sample 20% images from the training set as the validation set for architecture parameters updating. The original validation set is not used in the search process. The image is first resized to 512×1024 and 321×321 patches are randomly cropped as the input data. The stages where the expansion ratio of MBConv is 6 in the original MobileNetV2 are searched and adjusted. We set the numbers of layers in each searched stage of the super network as $\{4, 4, 6, 6, 4, 1\}$. To optimize the FLOPs of the searched network, we define the loss function in search as $\mathcal{L} = \mathcal{L}_{task} + \lambda \log_\tau(\gamma \cdot cost)$. The first term denotes the cross-entropy loss and the second term controls the FLOPs of the network. We set λ as 9×10^{-3}, τ as 45 and γ as 10. The search process takes 80 epochs in total. We set a warm-up stage in the first 5 epochs to linearly increase the learning rate from 1×10^{-4} to 0.02. Then, the learning rate decays to 1×10^{-3} with the cosine annealing schedule (Loshchilov & Hutter, 2017). We use the SGD optimizer with 0.9 momentum and 5×10^{-4} weight decay for operation weights and the Adam optimizer (Kingma & Ba, 2015) with 4×10^{-5} weight decay and a fixed learning rate 1×10^{-3}. The architecture optimization starts after 30 epochs. The batch size is set as 16. The whole search process is conducted on a single V100 GPU and takes only 1.4 hours in total.

In the parameters adaptation process, we remap the parameters of original MobileNetV2 to the inner network obtained in the aforementioned architecture adaptation. The training data is cropped as a 769×769 patch from the rescaled image. The scale is randomly selected from $[0.75, 1.0, 1.25, 1.5, 1.75, 2.0]$. The random left-right flipping is used. We update the statistics of

Table 1: Semantic segmentation results on Cityscapes. The result of DPC in the brackets is our implemented version under the same settings as FNA. The MAdds of the models are computed with the 1024×2048 input resolution.

Method		OS	iters	Params	MAdds	mIOU(%)
MobileNetV2 (Sandler et al., 2018)	DeepLabv3	16	100K	2.57M	24.52B	75.5
DPC (Chen et al., 2018a)				2.51M	24.69B	75.4(75.7)
FNA				2.55M	24.17B	**76.6**
Auto-DeepLab-S (Liu et al., 2019a)	DeepLabv3+	8	500K	10.15M	333.25B	75.2
FNA		16	100K	5.71M	210.11B	77.2
FNA		8	100K	5.71M	313.87B	**78.0**

Table 2: Comparison of computational cost on the semantic segmentation tasks. AA: Architecture Adaptation. PA: Parameters Adaptation. GHs: GPU Hours. $*$ indicates the computational cost calculated under our reproducing settings. † indicates the cost estimated according to the description in the original paper (Liu et al., 2019a).

Method	Total Cost	AA Cost	PA Cost
DPC (Chen et al., 2018a)	62.2K GHs	62.2K GHs	30.0* GHs
Auto-DeepLab-S (Liu et al., 2019a)	244.0 GHs	72.0 GHs	172.0† GHs
FNA	35.8 GHs	1.4 GHs	34.4 GHs

the batch normalization (BN) (Ioffe & Szegedy, 2015) for 2000 iterations before the fine-tuning process. We use the same SGD optimizer as the search process. The learning rate linearly increases from 1×10^{-4} to 0.01 and then decays to 0 with the polynomial schedule. The batch size is set as 16. The whole parameters adaptation process is conducted on 4 TITAN-Xp GPUs and takes 100K iterations, which cost only 8.5 hours in total.

Our semantic segmentation results are shown in Tab. 1. The FNA network achieves 76.6% mIOU on Cityscapes with DeepLabv3 (Chen et al., 2017b) framework, 1.1% better than the manually designed seed Network MobileNetV2 (Sandler et al., 2018) with fewer MAdds. Compared to a NAS method DPC (Chen et al., 2018a) (with MobileNetV2 as the backbone) which searches the multi-scale module for semantic segmentation tasks, FNA gets 0.9% mIOU promotion with 0.52B fewer MAdds. For fair comparison with Auto-DeepLab (Liu et al., 2019a) which searches the backbone architecture on DeepLabv3 and retrains the searched network on DeepLabv3+ (Chen et al., 2018b), we adapt the parameters of the inner network $\mathcal{N}_i$ to DeepLabv3+ framework. Comparing to Auto-DeepLab-S, FNA achieves far better mIOU with fewer MAdds, Params and training iterations. With the remapping mechanism, FNA only takes 35.8 GPU hours, $1737\times$ less than DPC (Chen et al., 2018a) and $6.8\times$ less than Auto-DeepLab (Liu et al., 2019a).

4.2 Adaptation on Object Detection

We further implement our FNA method to object detection tasks. We adapt the MobileNetV2 seed network to two commonly used detection systems, RetinaNet (Lin et al., 2017) and a lightweight one SSDLite (Liu et al., 2016; Sandler et al., 2018). The experiments are conducted on MS-COCO dataset (Lin et al., 2014). Our implementation is based on the MMDetection (Chen et al., 2019a) framework. In the search process of architecture adaptation, we randomly sample 50% data from the original training set as the validation set. The depth settings in each searched stage are set as $\{4, 4, 4, 4, 4, 1\}$. We describe more implementation details in the Appendix.

We demonstrate the results on object detection tasks in Tab. 3. In the RetinaNet (Lin et al., 2017) framework, we compare FNA with two manually designed networks, ShuffleNetV2-10 (Ma et al., 2018; Chen et al., 2019b) and MobileNetV2 (Sandler et al., 2018). FNA achieves higher mAP with similar MAdds. We compare FNA with DetNAS (Chen et al., 2019b) which searches the backbone of detection network. FNA achieves 0.6% higher mAP with 1.64M fewer Params and 0.2B fewer MAdds. As shown in Tab. 4, our total computational cost is only 13.5% of DetNAS, while DetNAS requires a huge cost for ImageNet pre-training. We degrade the cost largely with the assistance of parameters remapping. Moreover, we implement our FNA method on SSDLite (Liu et al., 2016; Sandler et al., 2018) framework. In Tab. 3, FNA surpasses both the manually designed network MobileNetV2 and the NAS-searched network MnasNet-92, where MnasNet (Tan et al., 2018) takes around 3.8K GPU days to search for the backbone network on ImageNet. The specific cost FNA takes on SSDLite is shown in Tab. 4. It is difficult to train the small network due to the

Table 3: Object detection results on MS-COCO. The MAdds are calculated with 1088×800 input.

Method		Params	MAdds	mAP(%)
ShuffleNetV2-20 (Chen et al., 2019b)	RetinaNet	13.19M	132.76B	32.1
MobileNetV2 (Sandler et al., 2018)		11.49M	133.05B	32.8
DetNAS (Chen et al., 2019b)		13.41M	133.26B	33.3
FNA		11.74M	133.03B	**33.9**
MobileNetV2 (Sandler et al., 2018)	SSDLite	4.3M	0.8B	22.1
Mnasnet-92 (Tan et al., 2018)		5.3M	1.0B	22.9
FNA		4.6M	0.9B	**23.0**

Table 4: Comparison of computational cost on the object detection tasks. All our experiments on object detection are conducted on TITAN-Xp.

Method	Total Cost	Super Network			Target Network	
		Pre-training	Finetuning	Search	Pre-training	Finetuning
DetNAS (Chen et al., 2019b)	68 GDs	12 GDs	12 GDs	20 GDs	12 GDs	12 GDs
FNA (RetinaNet)	9.2 GDs	-	-	6 GDs	-	3.2 GDs
FNA (SSDLite)	21.6 GDs	-	-	6.6 GDs	-	15 GDs

simplification (Liu et al., 2019c). Therefore, experiments on SSDLite need long training schedule and take larger computational cost. The experimental results further demonstrate the efficiency and effectiveness of direct adaptation on the target task with parameters remapping.

4.3 Effectiveness of Parameters Remapping

To validate the effectiveness of the parameters remapping paradigm in our method, we carry out several experiments for ablation study. The experiments are conducted with the semantic segmentation framework DeepLabv3 (Chen et al., 2017b) on Cityscapes dataset (Cordts et al., 2016). In our FNA method, the parameters remapping is conducted on both stages of network expansion and parameters adaptation. To prove the importance of parameters remapping in FNA, we attempt to remove the parameters remapping process at the two stages. The parameters of the network without remapping will be initialized with random values based on the standard normal distribution. With parameters remapping applied on both stages, the adaptation achieves the best results in Tab. 5. Especially, the remapping operation on the parameters adaptation stage tends to have greater importance for the final performance.

To further evaluate the performance of parameters remapping on the parameters adaptation stage, we pre-train the inner network $\mathcal{N}_i$ obtained in architecture adaptation on ImageNet. Then we finetune the ImageNet pre-trained network on the semantic segmentation task. All the other settings in the experiments are the same as FNA. It is worth noting that FNA even achieves higher mIOU by a narrow margin (0.1%) in Tab. 5. We conjecture that this may benefit from the regularization of parameters remapping in the parameters adaptation stage.

Table 5: Study the effectiveness of parameters remapping. NE: Network Expansion. PA: Parameters Adaptation. PR: Parameters Remapping. ImageNet: pre-training on ImageNet. - denotes that the parameters of the network are randomly initialized.

NE	-	PR	-	PR	PR
PA	-	-	PR	ImageNet	PR
mIOU(%)	72.4	73.0	76.0	76.5	**76.6**

4.4 Study on Parameters Remapping

We explore more other methods of the parameters remapping paradigm. Similar to Sec. 4.3, all the experiments are conducted with DeepLabv3 (Chen et al., 2017b) framework on Cityscapes dataset (Cordts et al., 2016). We make exploration from the following respects. For simplicity, we denote the weights of the seed network and the new network on the remapping dimension (output / input channel), $\boldsymbol{W}_s = (\boldsymbol{W}_s^{(1)} \dots \boldsymbol{W}_s^{(p)})$ and $\boldsymbol{W}_n = (\boldsymbol{W}_n^{(1)} \dots \boldsymbol{W}_n^{(q)})$ where $q \leq p$.

Remapping with BN Statistics on Width-level We review the formulation of batch normalization (Ioffe & Szegedy, 2015) as follows,

$$y_i \leftarrow \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta, \tag{5}$$

Algorithm 1: Weights Remapping Function

Input: the seed weights $\boldsymbol{W}_s$ and the new weights $\boldsymbol{W}_n$, the reference vector $\boldsymbol{v}$

// get indices of topk values of the vector

$\boldsymbol{a} \leftarrow topk\text{-}indices(\boldsymbol{v}, k = q)$

// sort the indices

$sort(\boldsymbol{a})$

for $i \in 1, 2, \ldots, q$ **do**

  $\boldsymbol{W}_n^{(i)} = \boldsymbol{W}_s^{(\boldsymbol{a}^{(i)})}$

end

Output: $\boldsymbol{W}_n$ with remapped values

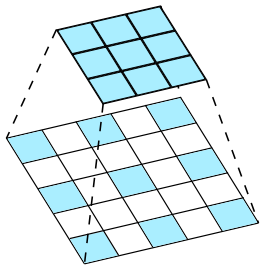


Figure 3: Parameters remapping on the kernel-level with a dilation setting.

where $x_i = (x_i^{(1)} \ldots x_i^{(p)})$ denotes the p-dimensional input tensor of the ith layer, $\gamma \in \mathbb{R}^p$ denotes the learnable parameter which scales the normalized data on the channel dimension. We compute the absolute values of γ as $|\gamma| = (|\gamma^{(1)}| \ldots |\gamma^{(p)}|)$. When remapping the parameters on the width-level, we sort the values of $|\gamma|$ and map the parameters with the sorted top-q indices. More specifically, we define a weights remapping function in Algo. 1, where the reference vector is $|\gamma|$.

Remapping with Weight Importance on Width-level We attempt to utilize a canonical form of convolution weights to measure the importance of parameters. Then we remap the seed network parameters with great importance to the new network. The remapping operation is conducted based on Algo. 1 as well. We experiment with two canonical forms of weights to compute the reference vector, the standard deviation of $\boldsymbol{W}_s$ as $(std(\boldsymbol{W}_s^{(1)}) \ldots std(\boldsymbol{W}_s^{(p)}))$ and the L^1 norm of $\boldsymbol{W}_s$ as $(||\boldsymbol{W}_s^{(1)}||_1 \ldots ||\boldsymbol{W}_s^{(p)}||_1)$.

Table 6: Study the methods of parameters remapping.

Method	Width-BN	Width-Std	Width-L1	Kernel-Dilate	FNA
mIOU(%)	75.8	75.8	75.3	75.6	**76.6**

Remapping with Dilation on Kernel-level We experiment with another method of parameters remapping on kernel-level. Different from the function-preserving method defined in Sec. 3.1, we remap the parameters with a dilation manner as shown in Fig. 7. The values in the convolution kernel without remapping are all assigned as 0. It is formulated as

$$\boldsymbol{W}_{:,:,h,w}^{k\times k} = \begin{cases} \boldsymbol{W}_{:,:,h,w}^{3\times 3} & \text{if } h, w = 1 + i \cdot \frac{k-1}{2} \text{ and } i = 0, 1, 2 \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

where $\boldsymbol{W}^{k\times k}$ and $\boldsymbol{W}^{3\times 3}$ denote the weights of the new network and the seed network, h, w denote the spatial indices.

Tab. 6 shows the experimental results. The network adaptation with the parameters remapping paradigm define in FNA achieves the best results. Furthermore, the remapping operation of FNA is easier to implement compared to the several aforementioned ones. However, we explore limited numbers of methods to implement the parameters-remapping paradigm. How to conduct the remapping strategy remains to be a significative future work.

5 Conclusion

In this paper, we propose a fast neural network adaptation method (FNA) with a parameters remapping paradigm. We adapt the manually designed network MobileNetV2 to semantic segmentation and detection tasks on both architecture- and parameter- level. The parameters-remapping strategy takes full advantage of the seed network parameters, which accelerates both the architecture search and parameters fine-tuning process greatly. With our FNA method, researchers and engineers could fast adapt more manually designed networks to the backbone of various frameworks on different tasks. As there are lots of ImageNet pre-trained models available in the community, we could conduct adaptation with little cost and do more applications, e.g., face recognition, pose estimation, depth estimation, etc. We leave more efficient remapping strategies and more applications for future work.

REFERENCES

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018.

Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. *arXiv:1708.05344*, 2017.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

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, and Dahua Lin. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019a.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017a.

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

Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *NeurIPS*, 2018a.

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018b.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv:1511.05641*, 2015.

Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv:1903.10979*, 2019b.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

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

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *ICLR*, 2019.

Jiemin Fang, Yukang Chen, Xinbang Zhang, Qian Zhang, Chang Huang, Gaofeng Meng, Wenyu Liu, and Xinggang Wang. EAT-NAS: elastic architecture transfer for accelerating large-scale neural architecture search. *arXiv:1901.05884*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

Kaiming He, Ross B. Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *arXiv:1811.08883*, 2018.

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:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: Design backbone for object detection. In *ECCV*, 2018.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, 2019a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019b.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

Zili Liu, Tu Zheng, Guodong Xu, Zheng Yang, Haifeng Liu, and Deng Cai. Training-time-friendly network for real-time object detection. *arXiv:1909.00700*, 2019c.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *arXiv:abs/1802.01548*, 2018.

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *NeurIPS*, 2015.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *LNCS*, 2015.

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*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv:1807.11626*, 2018.

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *arXiv:1908.07919*, 2019.

Robert J Wang, Xiang Li, and Charles X Ling. Pelee: A real-time object detection system on mobile devices. In *NeurIPS*, 2018.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *arXiv:1812.03443*, 2018.

Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, 2018.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

Yiheng Zhang, Zhaofan Qiu, Jingen Liu, Ting Yao, Dong Liu, and Tao Mei. Customizable architecture search for semantic segmentation. In *CVPR*, 2019.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.

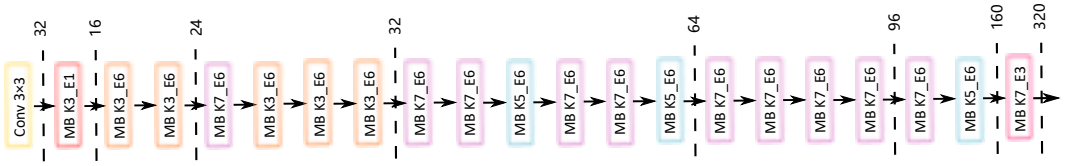
Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *arXiv:1707.07012*, 2017.

A Appendix

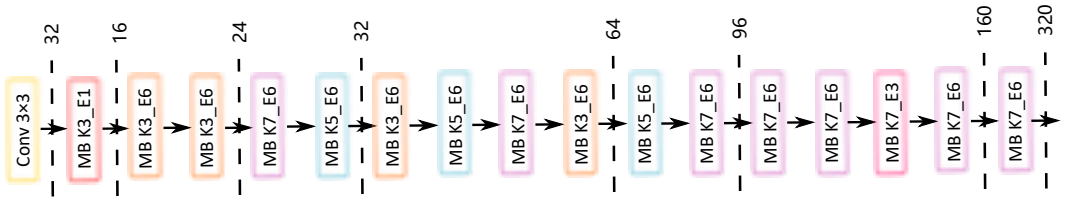
A.1 Implementation Details on Object Detection

RetinaNet We describe the details in the search process of architecture adaptation. For the input image size, the short side is resized to 800 while the maximum long side is set as 1333. For operation weights, we use the SGD optimizer with 1×10^{-4} weight decay and 0.9 momentum. We set a warm-up stage in the first 500 iterations to linearly increase the learning rate from 0 to 0.02. Then we decay the learning rate by a factor of 0.1 at 8 and 11 epochs. For the architecture parameters, we use the Adam optimizer (Kingma & Ba, 2015) with 1×10^{-3} weight decay and a fixed learning rate 3×10^{-4}. For the multi-objective loss function, we set λ as 0.02, τ as 10 and γ as 5. We begin optimizing the architecture parameters at 8 epochs. We remove the random flipping operation on input images in the search process. All the other training settings are the same as MMDetection (Chen et al., 2019a) implementation. For fine-tuning of the parameters adaptation, we use the SGD optimizer with 5×10^{-5} weight decay and 0.9 momentum. A similar warm-up procedure is set in the first 500 iterations to increase the learning rate from 0 to 0.05. Then we decay the learning rate by 0.1 at the 8 and 11 epochs. The whole architecture search process takes 14 epochs, 18 hours on 8 TITAN-Xp GPUs and the whole parameters fine-tuning takes 10 hours on 8 TITAN-Xp GPUs.

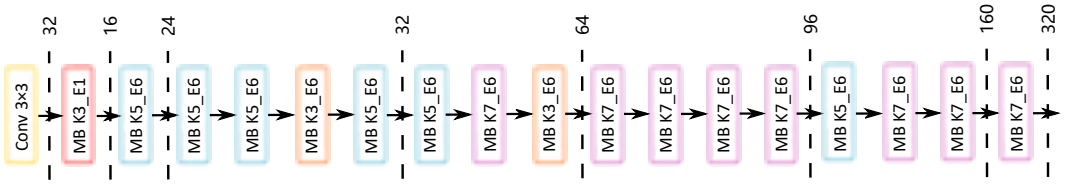
SSDLite We resize the input images to 320×320. For operation weights in the search process, we use the standard RMSProp optimizer with 4×10^{-5} weight decay. The warm-up stage in the first 500 iterations increases learning rate from 0 to 0.03. Then we decay the learning rate by 0.1 at 16 and 22 epochs. The architecture optimization starts at 12 epochs. We set λ as 0.2, τ as 10 and γ as 10 for the loss function. The other search settings are the same as the RetinaNet experiment. For parameters adaptation, the initial learning rate is 0.2 and decays at 36, 50 and 56 epochs. The training settings follow the SSD (Liu et al., 2016) implementation in MMDetection (Chen et al., 2019a). The search process takes 24 epochs in total, 20 hours on 8 TITAN-Xp GPUs. The parameters adaptation takes 60 epochs, 46 hours on 8 TITAN-Xp GPUs.



(a) FNA on DeepLabv3

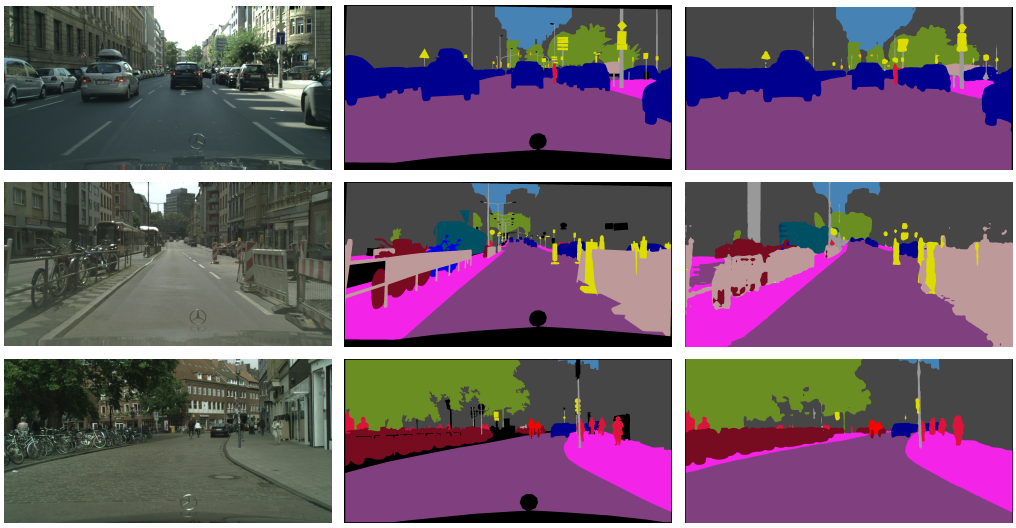


(b) FNA on RetinaNet



(c) FNA on SSDLite

Figure 4: Visualization of our searched architectures on different frameworks. MB: inverted residual block proposed in MobilenetV2 (Sandler et al., 2018). Kx_Ey: the kernel size of the depthwise convolution is x and the expansion ratio is y.



(a) Input (b) GT (c) Ours

Figure 5: Visualization of semantic segmentation results on the Cityscapes validation dataset.

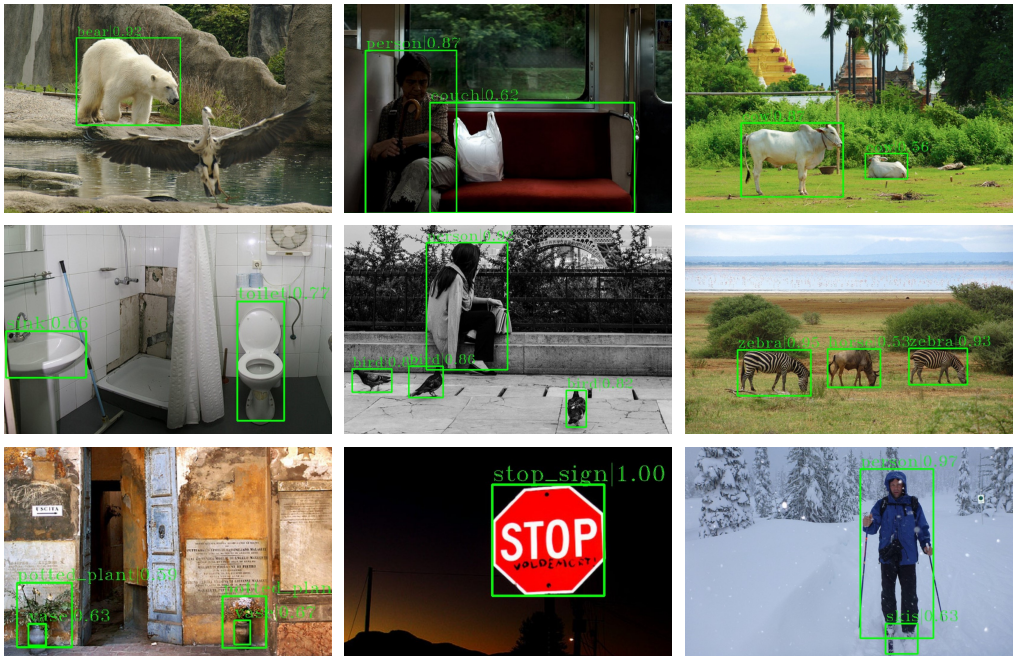


Figure 6: Visualization of object detection results on the MS-COCO validation dataset.