SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization

Xianzhi Du Tsung-Yi Lin Pengchong Jin Golnaz Ghiasi
Mingxing Tan Yin Cui Quoc V. Le Xiaodan Song
Google Research, Brain Team
{xianzhi,tsungyi,pengchong,golnazg,tanmingxing,yincui,qvl,xiaodansong}@google.com

Abstract

*Convolutional neural networks typically encode an input image into a series of intermediate features with decreasing resolutions. While this structure is suited to classification tasks, it does not perform well for tasks requiring simultaneous recognition and localization (*e.g.*, object detection). The encoder-decoder architectures are proposed to resolve this by applying a decoder network onto a backbone model designed for classification tasks. In this paper, we argue encoder-decoder architecture is ineffective in generating strong multi-scale features because of the scale-decreased backbone. We propose SpineNet, a backbone with scale-permuted intermediate features and cross-scale connections that is learned on an object detection task by Neural Architecture Search. Using similar building blocks, SpineNet models outperform ResNet-FPN models by ∼3% AP at various scales while using 10-20% fewer FLOPs. In particular, SpineNet-190 achieves* ***52.5% AP*** *with a Mask R-CNN detector and achieves* ***52.1% AP*** *with a RetinaNet detector on COCO for a single model without test-time augmentation, significantly outperforms prior art of detectors. SpineNet can transfer to classification tasks, achieving 5% top-1 accuracy improvement on a challenging iNaturalist fine-grained dataset. Code is at:* https://github.com/tensorflow/tpu/tree/master/models/official/detection.

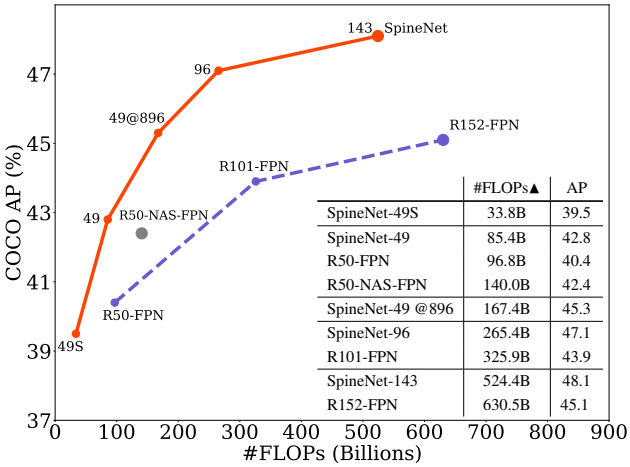


Figure 1: The comparison of RetinaNet models adopting SpineNet, ResNet-FPN, and NAS-FPN backbones. Details of training setup is described in Section 5 and controlled experiments can be found in Table 2, 3.

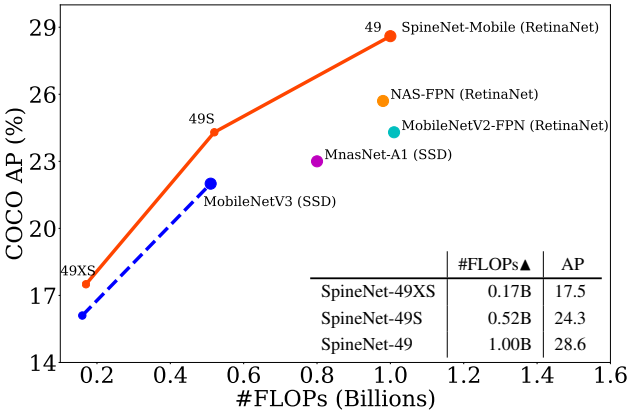


Figure 2: A comparison of mobile-size SpineNet models and other prior art of detectors for mobile-size object detection. Details are in Table 9.

1. Introduction

In the past a few years, we have witnessed a remarkable progress in deep convolutional neural network design. Despite networks getting more powerful by increasing depth and width [10, 43], the meta-architecture design has not been changed since the invention of convolutional neural networks. Most networks follow the design that encodes input image into intermediate features with monotonically decreased resolutions. Most improvements of network architecture design are in adding network depth and connections within feature resolution groups [19, 10, 14, 45]. LeCun *et al*. [19] explains the motivation behind this scale-decreased

architecture design: *"High resolution may be needed to detect the presence of a feature, while its exact position need not to be determined with equally high precision."*

The scale-decreased model, however, may not be able to deliver strong features for multi-scale visual recognition tasks where recognition and localization are both important (*e.g.*, object detection and segmentation). Lin et al. [21] shows directly using the top-level features from a scale-decreased model does not perform well on detecting small objects due to the low feature resolution. Several work including [21, 1] proposes multi-scale encoder-decoder architectures to address this issue. A scale-decreased network is taken as the encoder, which is commonly referred to a *backbone* model. Then a decoder network is applied to the backbone to recover the feature resolutions. The design of decoder network is drastically different from backbone model. A typical decoder network consists of a series of cross-scales connections that combine low-level and high-level features from a backbone to generate strong multi-scale feature maps. Typically, a backbone model has more parameters and computation (*e.g.*, ResNets [10]) than a decoder model (*e.g.*, feature pyramid networks [21]). Increasing the size of backbone model while keeping the decoder the same is a common strategy to obtain stronger encoder-decoder model.

In this paper, we aim to answer the question: Is the scale-decreased model a good backbone architecture design for simultaneous recognition and localization? Intuitively, a scale-decreased backbone throws away the spatial information by down-sampling, making it challenging to recover by a decoder network. In light of this, we propose a meta-architecture, called scale-permuted model, with two major improvements on backbone architecture design. First, the scales of intermediate feature maps should be able to increase or decrease anytime so that the model can retain spatial information as it grows deeper. Second, the connections between feature maps should be able to go across feature scales to facilitate multi-scale feature fusion. Figure 3 demonstrates the differences between scale-decreased and scale-permuted networks.

Although we have a simple meta-architecture design in mind, the possible instantiations grow combinatorially with the model depth. To avoid manually sifting through the tremendous amounts of design choices, we leverage Neural Architecture Search (NAS) [44] to learn the architecture. The backbone model is learned on the object detection task in COCO dataset [23], which requires simultaneous recognition and localization. Inspired by the recent success of NAS-FPN [6], we use the simple one-stage RetinaNet detector [22] in our experiments. In contrast to learning feature pyramid networks in NAS-FPN, we learn the backbone model architecture and directly connect it to the following classification and bounding box regression subnets. In other words, we remove the distinction between backbone and decoder models. The whole backbone model can be viewed and used as a feature pyramid network.

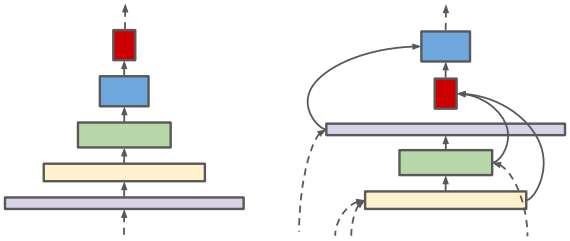


Figure 3: An example of scale-decreased network (left) *vs*. scale-permuted network (right). The width of block indicates feature resolution and the height indicates feature dimension. Dotted arrows represent connections from/to blocks not plotted.

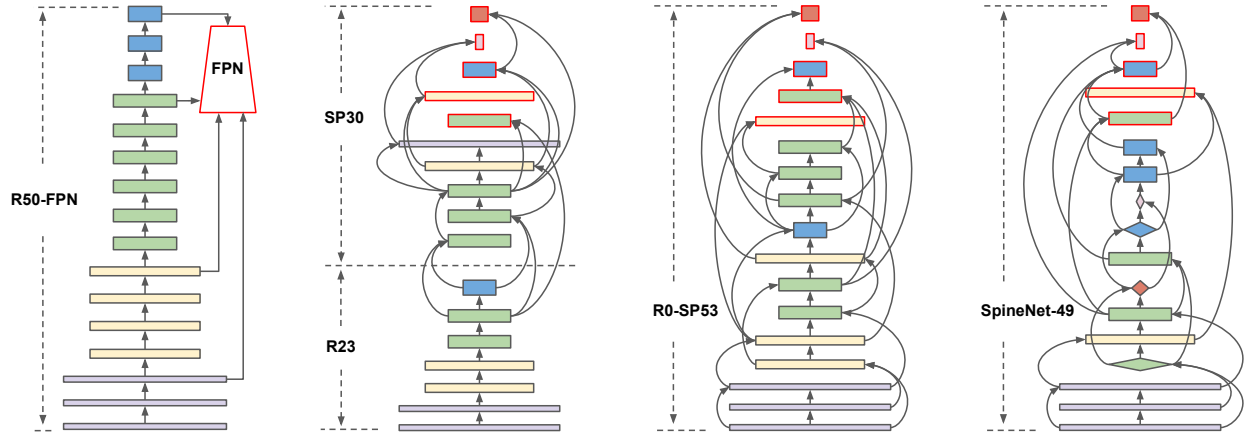
Taking ResNet-50 [10] backbone as our baseline, we use the bottleneck blocks in ResNet-50 as the candidate feature blocks in our search space. We learn (1) the permutations of feature blocks and (2) the two input connections for each feature block. All candidate models in the search space have roughly the same computation as ResNet-50 since we just permute the ordering of feature blocks to obtain candidate models. The learned scale-permuted model outperforms ResNet-50-FPN by (*+2.9% AP*) in the object detection task. The efficiency can be further improved (*-10% FLOPs*) by adding search options to adjust scale and type (*e.g.*, residual block or bottleneck block) of each candidate feature block. We name the learned scale-permuted backbone architecture SpineNet. Extensive experiments demonstrate that scale permutation and cross-scale connections are critical for building a strong backbone model for object detection. Figure 1 shows comprehensive comparisons of SpineNet to recent work in object detection.

We further evaluate SpineNet on ImageNet and iNaturalist classification datasets. Even though SpineNet architecture is learned with object detection, it transfers well to classification tasks. Particularly, SpineNet outperforms ResNet by 5% top-1 accuracy on iNaturalist fine-grained classification dataset, where the classes need to be distinguished with subtle visual differences and localized features. The ability of directly applying SpineNet to classification tasks shows that the scale-permuted backbone is versatile and has the potential to become a unified model architecture for many visual recognition tasks.

2. Related Work

2.1. Backbone Model

The progress of developing convolutional neural networks has mainly been demonstrated on ImageNet classification dataset [4]. Researchers have been improving model by increasing network depth [18], novel network connec-



(a) R50-FPN @37.8% AP (b) R23-SP30 @39.6% AP (c) R0-SP53 @40.7% AP (d) SpineNet-49 @40.8% AP

Figure 4: **Building scale-permuted network by permuting ResNet.** From (a) to (d), the computation gradually shifts from ResNet-FPN to scale-permuted networks. (a) The R50-FPN model, spending most computation in ResNet-50 followed by a FPN, achieves 37.8% AP; (b) R23-SP30, investing 7 blocks in a ResNet and 10 blocks in a scale-permuted network, achieves 39.6% AP; (c) R0-SP53, investing all blocks in a scale-permuted network, achieves 40.7% AP; (d) The SpineNet-49 architecture achieves 40.8% AP with 10% fewer FLOPs (85.4B *vs*. 95.2B) by learning additional block adjustments. Rectangle block represent bottleneck block and diamond block represent residual block. Output blocks are indicated by red border.

tions [10, 35, 36, 34, 14, 13], enhancing model capacity [43, 17] and efficiency [3, 32, 12, 38]. Several works have demonstrated that using a model with higher ImageNet accuracy as the backbone model achieves higher accuracy in other visual prediction tasks [16, 21, 1].

However, the backbones developed for ImageNet may not be effective for localization tasks, even combined with a decoder network such as [21, 1]. DetNet [20] argues that down-sampling features compromises its localization capability. HRNet [40] attempts to address the problem by adding parallel multi-scale inter-connected branches. Stacked Hourglass [27] and FishNet [33] propose recurrent down-sample and up-sample architecture with skip connections. Unlike backbones developed for ImageNet, which are mostly scale-decreased, several works above have considered backbones built with both down-sample and up-sample operations. In Section 5.5 we compare the scale-permuted model with Hourglass and Fish shape architectures.

2.2. Neural Architecture Search

Neural Architecture Search (NAS) has shown improvements over handcrafted models on image classification in the past few years [45, 25, 26, 41, 29, 38]. Unlike handcrafted networks, NAS learns architectures in the given search space by optimizing the specified rewards. Recent work has applied NAS for vision tasks beyond classification. NAS-FPN [6] and Auto-FPN [42] are pioneering works to apply NAS for object detection and focus on learning multi-layer feature pyramid networks. DetNAS [2] learns the backbone model and combines it with standard FPN [21]. Besides object detection, Auto-DeepLab [24] learns the backbone model and combines it with decoder in DeepLabV3 [1] for semantic segmentation. All aforementioned works except Auto-DeepLab learn or use a scale-decreased backbone model for visual recognition.

3. Method

The architecture of the proposed backbone model consists of a fixed stem network followed by a learned scale-permuted network. A stem network is designed with scale-decreased architecture. Blocks in the stem network can be candidate inputs for the following scale-permuted network.

A scale-permuted network is built with a list of building blocks $\{\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_N\}$. Each block $\mathbf{B}_k$ has an associated feature level L_i. Feature maps in an L_i block have a resolution of $\frac{1}{2^i}$ of the input resolution. The blocks in the same level have an identical architecture. Inspired by NAS-FPN [6], we define 5 output blocks from L_3 to L_7 and a 1×1 convolution attached to each output block to produce multi-scale features P_3 to P_7 with the same feature dimension. The rest of the building blocks are used as intermediate blocks before the output blocks. In Neural Architecture Search, we first search for scale permutations for the intermediate and output blocks then determine cross-scale connections between blocks. We further improve the model by adding block adjustments in the search space.

3.1. Search Space

Scale permutations: The orderings of blocks are important because a block can only connect to its parent blocks which have lower orderings. We define the search space of scale permutations by permuting intermediate and output blocks respectively, resulting in a search space size of $(N-5)!5!$. The scale permutations are first determined before searching for the rest of the architecture.

Cross-scale connections: We define two input connections for each block in the search space. The parent blocks can be any block with a lower ordering or block from the stem network. Resampling spatial and feature dimensions is needed when connecting blocks in different feature levels. The search space has a size of $\prod_{i=m}^{N+m-1} C_2^i$, where m is the number of candidate blocks in the stem network.

Block adjustments: We allow block to adjust its scale level and type. The intermediate blocks can adjust levels by $\{-1, 0, 1, 2\}$, resulting in a search space size of 4^{N-5}. All blocks are allowed to select one between the two options $\{$*bottleneck block*, *residual block*$\}$ described in [10], resulting in a search space size of 2^N.

3.2. Resampling in Cross-scale Connections

One challenge in cross-scale feature fusion is that the resolution and feature dimension may be different among parent and target blocks. In such case, we perform spatial and feature resampling to match the resolution and feature dimension to the target block, as shown in detail in Figure 5. Here, C is the feature dimension of 3×3 convolution in residual or bottleneck block. We use C^{in} and C^{out} to indicate the input and output dimension of a block. For bottleneck block, $C^{in} = C^{out} = 4C$; and for residual block, $C^{in} = C^{out} = C$. As it is important to keep the computational cost in resampling low, we introduce a scaling factor α (default value 0.5) to adjust the output feature dimension C^{out} in a parent block to αC. Then, we use a nearest-neighbor interpolation for up-sampling or a stride-2 3×3 convolution (followed by stride-2 max poolings if necessary) for down-sampling feature map to match to the target resolution. Finally, a 1×1 convolution is applied to match feature dimension αC to the target feature dimension C^{in}. Following FPN [21], we merge the two resampled input feature maps with elemental-wise addition.

3.3. Scale-Permuted Model by Permuting ResNet

Here we build scale-permuted models by permuting feature blocks in ResNet architecture. The idea is to have a fair comparison between scale-permuted model and scale-decreased model when using the same building blocks. We make small adaptation for scale-permuted models to generate multi-scale outputs by replacing one L_5 block in ResNet with one L_6 and one L_7 blocks and set the feature dimension to 256 for L_5, L_6, and L_7 blocks. In addition to comparing fully scale-decreased and scale-permuted model, we create a family of models that gradually shifts the model from the scale-decreased stem network to the scale-permuted network. Table 1 shows an overview of block allocation of models in the family. We use R[N]-SP[M] to indicate N feature layers in the handcrafted stem network and M feature layers in the learned scale-permuted network.

For a fair comparison, we constrain the search space to only include scale permutations and cross-scale connections. Then we use reinforcement learning to train a controller to generate model architectures. similar to [6], for intermediate blocks that do not connect to any block with a higher ordering in the generated architecture, we connect them to the output block at the corresponding level. Note that the cross-scale connections only introduce small computation overhead, as discussed in Section 3.2. As a result, all models in the family have similar computation as ResNet-50. Figure 4 shows a selection of learned model architectures in the family.

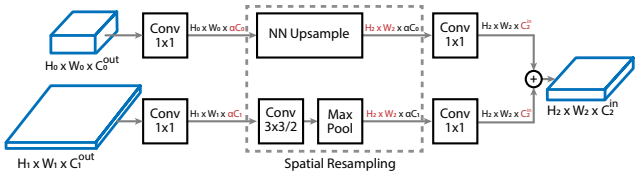


Figure 5: **Resampling operations.** Spatial resampling to upsample (top) and to downsample (bottom) input features followed by resampling in feature dimension before feature fusion.

	stem network $\{L_2, L_3, L_4, L_5\}$	scale-permuted network $\{L_2, L_3, L_4, L_5, L_6, L_7\}$
R50	$\{3, 4, 6, 3\}$	$\{-\}$
R35-SP18	$\{2, 3, 5, 1\}$	$\{1, 1, 1, 1, 1, 1\}$
R23-SP30	$\{2, 2, 2, 1\}$	$\{1, 2, 4, 1, 1, 1\}$
R14-SP39	$\{1, 1, 1, 1\}$	$\{2, 3, 5, 1, 1, 1\}$
R0-SP53	$\{2, 0, 0, 0\}$	$\{1, 4, 6, 2, 1, 1\}$
SpineNet-49	$\{2, 0, 0, 0\}$	$\{1, 2, 4, 4, 2, 2\}$

Table 1: **Number of blocks per level for stem and scale-permuted networks.** The scale-permuted network is built on top of a scale-decreased stem network as shown in Figure 4. The size of scale-decreased stem network is gradually decreased to show the effectiveness of scale-permuted network.

3.4. SpineNet Architectures

To this end, we design scale-permuted models with a fair comparison to ResNet. However, using ResNet-50 building blocks may not be an optimal choice for building scale-permuted models. We suspect the optimal model may have different feature resolution and block type distributions than

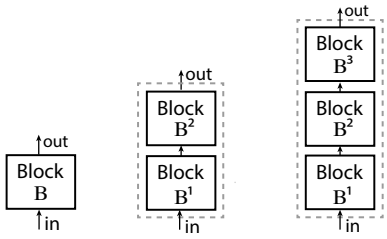


Figure 6: **Increase model depth by block repeat.** From left to right: blocks in SpineNet-49, SpineNet-96, and SpineNet-143.

ResNet. Therefore, we further include additional block adjustments in the search space as proposed in Section 3.1. The learned model architecture is named SpineNet-49, of which the architecture is shown in Figure 4d and the number of blocks per level is given in Table 1.

Based on SpineNet-49, we construct four architectures in the SpineNet family where the models perform well for a wide range of latency-performance trade-offs. The models are denoted as SpineNet-49S/96/143/190: SpineNet-49S has the same architecture as SpineNet-49 but the feature dimensions in the entire network are scaled down uniformly by a factor of 0.65. SpineNet-96 doubles the model size by repeating each block $\mathbf{B}_k$ twice. The building block $\mathbf{B}_k$ is duplicated into $\mathbf{B}_k^1$ and $\mathbf{B}_k^2$, which are then sequentially connected. The first block $\mathbf{B}_k^1$ connects to input parent blocks and the last block $\mathbf{B}_k^2$ connects to output target blocks. SpineNet-143 and SpineNet-190 repeat each block 3 and 4 times to grow the model depth and adjust α in the resampling operation to 1.0. SpineNet-190 further scales up feature dimension uniformly by 1.3. Figure 6 shows an example of increasing model depth by repeating blocks.

Note we do not apply recent work on new building blocks (*e.g.*, ShuffleNetv2 block used in DetNas [2]) or efficient model scaling [38] to SpineNet. These improvements could be orthogonal to this work.

4. Applications

4.1. Object Detection

The SpineNet architecture is learned with RetinaNet detector by simply replacing the default ResNet-FPN backbone model. To employ SpineNet in RetinaNet, we follow the architecture design for the class and box subnets in [22]: For SpineNet-49S, we use 4 shared convolutional layers at feature dimension 128; For SpineNet-49/96/143, we use 4 shared convolutional layers at feature dimension 256; For SpineNet-190, we scale up subnets by using 7 shared convolutional layers at feature dimension 512. To employ SpineNet in Mask R-CNN, we follow the same architecture design in [8]: For SpineNet-49S/49/96/143, we use 1 shared convolutional layers at feature dimension 256 for RPN, 4 shared convolutional layers at feature dimension 256 followed by a fully-connected layers of 1024 units for detection branch, and 4 shared convolutional layers at feature dimension 256 for mask branch. For SpineNet-49S, we use 128 feature dimension for convolutional layers in subnets. For SpineNet-190, we scale up detection subnets by using 7 convolutional layers at feature dimension 384.

4.2. Image Classification

To demonstrate SpineNet has the potential to generalize to other visual recognition tasks, we apply SpineNet to image classification. We utilize the same P_3 to P_7 feature pyramid to construct the classification network. Specifically, the final feature map $P = \frac{1}{5}\sum_{i=3}^{7}\mathcal{U}(P_i)$ is generated by upsampling and averaging the feature maps, where $\mathcal{U}(\cdot)$ is the nearest-neighbor upsampling to ensure all feature maps have the same scale as the largest feature map P_3. The standard global average pooling on P is applied to produce a 256-dimensional feature vector followed by a linear classifier with softmax for classification.

5. Experiments

For object detection, we evaluate SpineNet on COCO dataset [23]. All the models are trained on the `train2017` split. We report our main results with COCO AP on the `test-dev` split and others on the `val2017` split. For image classification, we train SpineNet on ImageNet ILSVRC-2012 [31] and iNaturalist-2017 [39] and report Top-1 and Top-5 validation accuracy.

5.1. Experimental Settings

Training data pre-processing: For object detection, we feed a larger image, from 640 to 896, 1024, 1280, to a larger SpineNet. The long side of an image is resized to the target size then the short side is padded with zeros to make a square image. For image classification, we use the standard input size of 224 × 224. During training, we adopt standard data augmentation (scale and aspect ratio augmentation, random cropping and horizontal flipping).

Training details: For object detection, we generally follow [22, 6] to adopt the same training protocol, denoting as protocol A, to train SpineNet and ResNet-FPN models for controlled experiments described in Figure 4. In brief, we use stochastic gradient descent to train on Cloud TPU v3 devices with 4e-5 weight decay and 0.9 momentum. All models are *trained from scratch* on COCO `train2017` with 256 batch size for 250 epochs. The initial learning rate is set to 0.28 and a linear warmup is applied in the first 5 epochs. We apply stepwise learning rate that decays to $0.1\times$ and $0.01\times$ at the last 30 and 10 epoch. We follow [8] to apply synchronized batch normalization with 0.99 momentum followed by ReLU and implement DropBlock [5] for regularization. We apply multi-scale training with a random

backbone model	resolution	#FLOPs▲	#Params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SpineNet-49S	640×640	**33.8B**	**11.9M**	**39.5**	**59.3**	**43.1**	**20.9**	**42.2**	**54.3**
SpineNet-49	640×640	**85.4B**	**28.5M**	**42.8**	**62.3**	**46.1**	**23.7**	**45.2**	**57.3**
R50-FPN	640×640	96.8B	34.0M	40.4	59.9	43.6	22.7	43.5	57.0
R50-NAS-FPN	640×640	140.0B	60.3M	42.4	61.8	46.1	25.1	46.7	57.8
SpineNet-49	896×896	**167.4B**	**28.5M**	**45.3**	**65.1**	**49.1**	**27.0**	**47.9**	**57.7**
SpineNet-96	1024×1024	**265.4B**	**43.0M**	**47.1**	**67.1**	**51.1**	**29.1**	**50.2**	**59.0**
R101-FPN	1024×1024	325.9B	53.1M	43.9	63.6	47.6	26.8	47.6	57.0
SpineNet-143	1280×1280	**524.4B**	**66.9M**	**48.1**	**67.6**	**52.0**	**30.2**	**51.1**	**59.9**
R152-FPN	1280×1280	630.5B	68.7M	45.1	64.6	48.7	28.4	48.8	58.2
R50-FPN†	640×640	96.8B	34.0M	42.3	61.9	45.9	23.9	46.1	58.5
SpineNet-49S†	640×640	**33.8B**	**12.0M**	**41.5**	**60.5**	**44.6**	**23.3**	**45.0**	**58.0**
SpineNet-49†	640×640	**85.4B**	**28.5M**	**44.3**	**63.8**	**47.6**	**25.9**	**47.7**	**61.1**
SpineNet-49†	896×896	**167.4B**	**28.5M**	**46.7**	**66.3**	**50.6**	**29.1**	**50.1**	**61.7**
SpineNet-96†	1024×1024	**265.4B**	**43.0M**	**48.6**	**68.4**	**52.5**	**32.0**	**52.3**	**62.0**
SpineNet-143†	1280×1280	**524.4B**	**66.9M**	**50.7**	**70.4**	**54.9**	**33.6**	**53.9**	**62.1**
SpineNet-190†	1280×1280	**1885.0B**	**163.6M**	**52.1**	**71.8**	**56.5**	**35.4**	**55.0**	**63.6**

Table 2: **One-stage object detection results on COCO `test-dev`.** We compare employing different backbones with RetinaNet on single model without test-time augmentation. By default we apply protocol B with multi-scale training and ReLU activation to train all models in this table, as described in Section 5.1. Models marked by dagger (†) are trained with protocol C by applying stochastic depth and swish activation for a longer training schedule. FLOPs is represented by Multi-Adds.

model	block adju.	#FLOPs	AP
R50-FPN	-	96.8B	37.8
R35-SP18	-	91.7B	38.7
R23-SP30	-	96.5B	39.7
R14-SP39	-	99.7B	39.6
R0-SP53	-	95.2B	40.7
SpineNet-49	✓	85.4B	40.8

Table 3: Results comparisons between R50-FPN and scale-permuted models on COCO `val2017` by adopting protocol A. The performance improves with more computation being allocated to scale-permuted network. We also show the efficiency improvement by having scale and block type adjustments in Section 3.1.

model	resolution	AP	inference latency
SpineNet-49S	640×640	39.9	11.7ms
SpineNet-49	640×640	42.8	15.3ms
SpineNet-49	896×896	45.3	34.3ms

Table 4: Inference latency of RetinaNet with SpineNet on a V100 GPU with NVIDIA TensorRT. Latency is measured for an end-to-end object detection pipeline including pre-processing, detection generation, and post-processing (*e.g.*, NMS).

scale between $[0.8, 1.2]$ as in [6]. We set base anchor size to 3 for SpineNet-96 or smaller models and 4 for SpineNet-143 or larger models in RetinaNet implementation. For our reported results, we adopt an improved training protocol denoting as protocol B. For simplicity, protocol B removes DropBlock and apply stronger multi-scale training with a random scale between $[0.5, 2.0]$ for 350 epochs. To obtain the most competitive results, we add stochastic depth with keep prob 0.8 [15] for stronger regularization and replace ReLU with swish activation [28] to train all models for 500 epochs, denoting as protocol C. We also adopt a more aggressive multi-scale training strategy with a random scale between [0.1, 2.0] for SpineNet-143/190 when using protocol C. For image classification, all models are trained with a batch size of 4096 for 200 epochs. We used cosine learning rate decay [11] with linear scaling of learning rate and gradual warmup in the first 5 epochs [7].

NAS details: We implement the recurrent neural network based controller proposed in [44] for architecture search, as it is the only method we are aware of that supports searching for permutations. We reserve 7392 images from `train2017` as the validation set for searching. To speed up the searching process, we design a proxy SpineNet by uniformly scaling down the feature dimension of SpineNet-49 with a factor of 0.25, setting α in resampling to 0.25, and using feature dimension 64 in the box and class nets. To prevent the search space from growing exponentially, we restrict intermediate blocks to search for parent blocks within the last 5 blocks built and allow output blocks to search from all existing blocks. At each sample, a proxy task is trained at

backbone model	resolution	#FLOPs▲	#Params	AP_{val}	AP^{mask}_{val}	$AP_{test\text{-}dev}$	$AP^{mask}_{test\text{-}dev}$
SpineNet-49S	640×640	**60.2B**	**13.9M**	**39.3**	**34.8**	-	-
SpineNet-49	640×640	**216.1B**	**40.8M**	**42.9**	**38.1**	-	-
R50-FPN	640×640	227.7B	46.3M	42.7	37.8	-	-
SpineNet-96	1024×1024	**315.0B**	**55.2M**	**47.2**	**41.5**	-	-
R101-FPN	1024×1024	375.5B	65.3M	46.6	41.2	-	-
SpineNet-143	1280×1280	**498.8B**	**79.2M**	**48.8**	**42.7**	-	-
R152-FPN	1280×1280	605.3B	80.9M	48.1	42.4	-	-
SpineNet-190†	1536×1536	**2076.8B**	**176.2M**	**52.2**	**46.1**	**52.5**	**46.3**

Table 5: **Two-stage object detection and instance segmentation results.** We compare employing different backbones with Mask R-CNN using 1000 proposals on single model without test-time augmentation. By default we apply protocol B with multi-scale training and ReLU activation to train all models in this table, as described in Section 5.1. SpineNet-190 (marked by †) is trained with protocol C by applying stochastic depth and swish activation for a longer training schedule. FLOPs is represented by Multi-Adds.

image resolution 512 for 5 epochs. AP of the proxy task on the reserved validation set is collected as reward. The controller uses 100 Cloud TPU v3 in parallel to sample child models. The best architectures for R35-SP18, R23-SP30, R14-SP39, R0-SP53, and SpineNet-49 are found after 6k, 10k, 13k, 13k, and 14k architectures are sampled.

5.2. Learned Scale-Permuted Architectures

In Figure 4, we observe scale-permuted models have permutations such that the intermediate features undergo the transformations that constantly up-sample and down-sample feature maps, showing a big difference compared to a scale-decreased backbone. It is very common that two adjacent intermediate blocks are connected to form a *deep* pathway. The output blocks demonstrate a different behavior preferring longer range connections. In Section 5.5, we conduct ablation study to show the importance of learned scale permutation and connections.

5.3. ResNet-FPN vs. SpineNet

We first present the object detection results of the 4 scale-permuted models discussed in Section 3.3 and compare with the ResNet50-FPN baseline. The results in Table 3 support our claims that: (1) The scale-decreased backbone model is not a good design of backbone model for object detection; (2) allocating computation on the proposed scale-permuted model yields higher performance.

Compared to the R50-FPN baseline, R0-SP53 uses similar building blocks and gains 2.9% AP with a learned scale permutations and cross-scale connections. The SpineNet-49 model further improves efficiency by reducing FLOPs by 10% while achieving the same accuracy as R0-SP53 by adding scale and block type adjustments.

5.4. Object Detection Results

RetinaNet: We evaluate SpineNet architectures on the COCO bounding box detection task with a RetinaNet detector. The results are summarized in Table 2. SpineNet models outperform other popular detectors by large margins, such as ResNet-FPN, and NAS-FPN at various model sizes in both accuracy and efficiency. Our largest SpineNet-190 achieves 52.1% AP on single model object detection without test-time augmentation.

Mask R-CNN: We also show results of Mask R-CNN models with different backbones for COCO instance segmentation task. Being consistent with RetinaNet results, SpineNet based models are able to achieve better AP and mask AP with smaller model size and less number of FLOPs. Note that SpineNet is learned on box detection with RetinaNet but works well with Mask R-CNN.

Real-time Object Detection: Our SpineNet-49S and SpineNet-49 with RetinaNet run at 30+ fps with NVIDIA TensorRT on a V100 GPU. We measure inference latency using an end-to-end object detection pipeline including pre-processing, bounding box and class score generation, and post-processing with non-maximum suppression, reported in Table 4.

5.5. Ablation Studies

Importance of Scale Permutation: We study the importance of learning scale permutations by comparing learned scale permutations to fixed ordering feature scales. We choose two popular architecture shapes in encoder-decoder networks: (1) A *Hourglass* shape inspired by [27, 21]; (2) A *Fish* shape inspired by [33]. Table 7 shows the ordering of feature blocks in the *Hourglass* shape and the *Fish* shape architectures. Then, we learn cross-scale connections using the same search space described in Section 3.1. The performance shows jointly learning scale permutations and cross-scale connections is better than only learning connections with a fixed architecture shape. Note there may exist some architecture variants to make *Hourglass* and *Fish* shape model perform better, but we only experiment with one of the simplest fixed scale orderings.

network	ImageNet ILSVRC-2012 (1000-class)				iNaturalist-2017 (5089-class)			
	#FLOPs▲	#Params	Top-1 %	Top-5 %	#FLOPs	#Params	Top-1 %	Top-5 %
SpineNet-49	**3.5B**	**22.1M**	**77.0**	**93.3**	**3.5B**	**23.1M**	**59.3**	**81.9**
ResNet-34	3.7B	21.8M	74.4	92.0	3.7B	23.9M	54.1	76.7
ResNet-50	4.1B	25.6M	77.1	93.6	4.1B	33.9M	54.6	77.2
SpineNet-96	**5.7B**	**36.5M**	**78.2**	**94.0**	**5.7B**	**37.6M**	**61.7**	**83.4**
ResNet-101	7.8B	44.6M	78.2	94.2	7.8B	52.9M	57.0	79.3
SpineNet-143	**9.1B**	**60.5M**	**79.0**	**94.4**	**9.1B**	**61.6M**	**63.6**	**84.8**
ResNet-152	11.5B	60.2M	78.7	94.2	11.5B	68.6M	58.4	80.2

Table 6: **Image classification results on ImageNet and iNaturalist.** Networks are sorted by increasing number of FLOPs. Note that the penultimate layer in ResNet outputs a 2048-dimensional feature vector for the classifier while SpineNet's feature vector only has 256 dimensions. Therefore, on iNaturalist, ResNet and SpineNet have around 8M and 1M more parameters respectively.

model shape	fixed block ordering	AP
Hourglass	$\{3L_2, 3L_3, 5L_4, 1L_5, 1L_7, 1L_6, 1L_5, 1L_4, 1L_3\}$	38.3%
Fish	$\{2L_2, 2L_3, 3L_4, 1L_5, 2L_4, 1L_3, 1L_2, 1L_3, 1L_4, 1L_5, 1L_6, 1L_7\}$	37.5%
R0-SP53	-	40.7%

Table 7: **Importance of learned scale permutation.** We compare our R0-SP53 model to hourglass and fish models with fixed block orderings. All models learn the cross-scale connections by NAS.

model	long	short	sequential	AP
R0-SP53	✓	✓	-	40.7%
Graph damage (1)	✓	✗	-	35.8%
Graph damage (2)	✗	✓	-	28.6%
Graph damage (3)	✗	✗	✓	28.2%

Table 8: **Importance of learned cross-scale connections.** We quantify the importance of learned cross-scale connections by performing three graph damages by removing edges of: (1) short-range connections; (2) long-range connections; (3) all connections then sequentially connecting every pair of adjacent blocks.

Importance of Cross-scale Connections: The cross-scale connections play a crucial role in fusing features at different resolutions throughout a scale-permuted network. We study its importance by graph damage. For each block in the scale-permuted network of R0-SP53, cross-scale connections are damaged in three ways: (1) Removing the short-range connection; (2) Removing the long-range connection; (3) Removing both connections then connecting one block to its previous block via a sequential connection. In all three cases, one block only connects to one other block. In Table 8, we show scale-permuted network is sensitive to any of edge removal techniques proposed here. The (2) and (3) yield severer damage than (1), which is possibly because of short-range connection or sequential connection can not effectively handle the frequent resolution changes.

5.6. Image Classification with SpineNet

Table 6 shows the image classification results. Under the same setting, SpineNet's performance is on par with ResNet on ImageNet but using much fewer FLOPs. On iNaturalist, SpineNet outperforms ResNet by a large margin of around 5%. Note that iNaturalist-2017 is a challenging fine-grained classification dataset containing 579,184 training and 95,986 validation images from 5,089 classes.

To better understand the improvement on iNaturalist, we created iNaturalist-bbox with objects cropped by ground truth bounding boxes collected in [39]. The idea is to create a version of iNaturalist with an iconic single-scaled object centered at each image to better understand the performance improvement. Specifically, we cropped all available bounding boxes (we enlarge the cropping region to be 1.5× of the original bounding box width and height to include context around the object), resulted in 496,164 training and 48,736 validation images from 2,854 classes. On iNaturalist-bbox, the Top-1/Top-5 accuracy is 63.9%/86.9% for SpineNet-49 and 59.6%/83.3% for ResNet-50, with a 4.3% improvement in Top-1 accuracy. The improvement of SpineNet-49 over ResNet-50 in Top-1 is 4.7% on the original iNaturalist dataset. Based on the experiment, we believe the improvement on iNaturalist is not due to capturing objects of variant scales but the following 2 reasons: 1) capturing subtle local differences thanks to the multi-scale features in SpineNet; 2) more compact feature representation (256-dimension) that is less likely to overfit.

6. Conclusion

In this work, we identify that the conventional scale-decreased model, even with decoder network, is not effective for simultaneous recognition and localization. We propose the scale-permuted model, a new meta-architecture, to address the issue. To prove the effectiveness of scale-

backbone model	#FLOPs	#Params	AP	AP_S	AP_M	AP_L
SpineNet-49XS (MBConv)	**0.17B**	**0.82M**	**17.5**	**2.3**	**17.2**	**33.6**
MobileNetV3-Small-SSDLite [12]	0.16B	1.77M	16.1	-	-	-
SpineNet-49S (MBConv)	**0.52B**	**0.97M**	**24.3**	**7.2**	**26.2**	**41.1**
MobileNetV3-SSDLite [12]	0.51B	3.22M	22.0	-	-	-
MobileNetV2-SSDLite [32]	0.80B	4.30M	22.1	-	-	-
MnasNet-A1-SSDLite [37]	0.80B	4.90M	23.0	3.8	21.7	42.0
SpineNet-49 (MBConv)	**1.00B**	**2.32M**	**28.6**	**9.2**	**31.5**	**47.0**
MobileNetV2-NAS-FPNLite (7 @64) [6]	0.98B	2.62M	25.7	-	-	-
MobileNetV2-FPNLite [32]	1.01B	2.20M	24.3	-	-	-

Table 9: **Mobile-size object detection results.** We report single model results without test-time augmentation on COCO `test-dev`.

network	ImageNet ILSVRC-2012 (1000-class)				iNaturalist-2017 (5089-class)			
	#FLOPs▲	#Params	Top-1 %	Top-5 %	#FLOPs	#Params	Top-1 %	Top-5 %
SpineNet-49	3.5B	22.1M	77.0	93.3	3.5B	23.1M	59.3	81.9
SpineNet-49†			78.1	94.0			63.3	85.1
SpineNet-96	5.7B	36.5M	78.2	94.0	5.7B	37.6M	61.7	83.4
SpineNet-96†			79.4	94.6			64.7	85.9
SpineNet-143	9.1B	60.5M	79.0	94.4	9.1B	61.6M	63.6	84.8
SpineNet-143†			80.1	95.0			66.7	87.1
SpineNet-190†	19.1B	127.1M	80.8	95.3	19.1B	129.2M	67.6	87.4

Table 10: The performance of SpineNet classification model can be further improved with a better training protocol by 1) adding stochastic depth, 2) replacing ReLU with swish activation and 3) using label smoothing of 0.1 (marked by †).

permuted models, we learn SpineNet by Neural Architecture Search in object detection and demonstrate it can be used directly in image classification. SpineNet significantly outperforms prior art of detectors by achieving 52.1% AP on COCO `test-dev`. The same SpineNet architecture achieves a comparable top-1 accuracy on ImageNet with much fewer FLOPs and 5% top-1 accuracy improvement on challenging iNaturalist dataset. In the future, we hope the scale-permuted model will become the meta-architecture design of backbones across many visual tasks beyond detection and classification.

Acknowledgments: We would like to acknowledge Yeqing Li, Youlong Cheng, Jing Li, Jianwei Xie, Russell Power, Hongkun Yu, Chad Richards, Liang-Chieh Chen, Anelia Angelova, and the Google Brain team for their help.

Appendix A: Mobile-size Object Detection

For mobile-size object detection, we explore building SpineNet with MBConv blocks using the parametrization proposed in [37], which is the inverted bottleneck block [32] with SE module [13]. Following [37], we set feature dimension $\{16, 24, 40, 80, 112, 112, 112\}$, expansion ratio 6, and kernel size 3×3 for L_1 to L_7 MBConv blocks. Each block in SpineNet-49 is replaced with the MBConv block at the corresponding level. Similar to [37], we replace the first convolution and maxpooling in stem with a 3×3 convolution at feature dimension 8 and a L_1 MBConv block respectively and set the first L_2 block to stride 2. The first 1×1 convolution in resampling to adjust feature dimension is removed. All convolutional layers in resampling operations and box/class nets are replaced with separable convolution in order to have comparable computation with MBConv blocks. Feature dimension is reduced to 48 in the box/class nets. We further construct SpineNet-49XS and SpineNet-49S by scaling the feature dimension of SpineNet-49 by $0.6\times$ and $0.65\times$ and setting the feature dimension in the box/class nets to 24 and 40 respectively. We adopt training protocol B with swish activation to train all models with RetinaNet for 600 epochs at resolution 256 for SpineNet-49XS and 384 for other models. The results are presented in Table 9 and the FLOPs *vs.* AP curve is plotted in Figure 2. Bulit with MBConv blocks, SpineNet-49XS/49S/49 use less computation but outperform MnasNet, MobileNetV2, and MobileNetV3 by 2-4% AP.

Note that as all the models in this section use handcrafted

MBConv blocks, the performance should be no better than a joint search of SpineNet and MBConv blocks with NAS.

Appendix B: Image Classification

Inspired by protocol C, we conduct SpineNet classification experiments using an improved training protocol by 1) adding stochastic depth, 2) replacing ReLU with swish activation and 3) using label smoothing of 0.1. From results in Table 10, we can see that the improved training protocol yields around 1% Top-1 gain on ImageNet and 3-4% Top-1 gain on iNaturalist-2017.

References

[1] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 2, 3

[2] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. In *Advances in Neural Information Processing Systems*, 2019. 3, 5

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 3

[4] 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

[5] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, 2018. 6

[6] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 2, 3, 4, 5, 6, 9

[7] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 6

[8] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *ICCV*, 2019. 5

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

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 3, 4

[11] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019. 6

[12] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 3, 9

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 3, 9

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1, 3

[15] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 6

[16] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*, 2017. 3

[17] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018. 3

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 2

[19] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. 1

[20] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: Design backbone for object detection. In *ECCV*, 2018. 3

[21] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 2, 3, 4, 7

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 2, 5

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 2, 5

[24] 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*, 2019. 3

[25] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 3

[26] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2018. 3

[27] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016. 3, 7

[28] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017. 6

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 3

[30] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 5

[32] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 3, 9

[33] Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. In *Advances in Neural Information Processing Systems*, 2018. 3, 7

[34] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 3

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 3

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 3

[37] 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 *CVPR*, 2019. 9

[38] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 3, 5

[39] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *CVPR*, 2018. 5, 8

[40] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *PAMI*, 2020. 3

[41] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *ICCV*, 2019. 3

[42] Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *ICCV*, 2019. 3

[43] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 1, 3

[44] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2, 6

[45] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 3