SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization

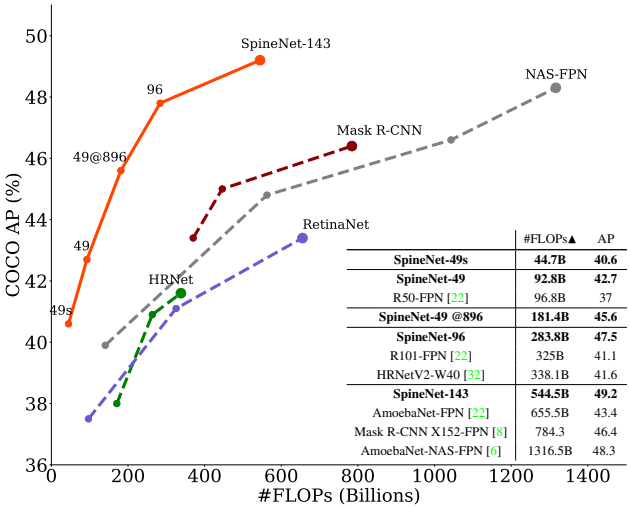
Xianzhi Du Tsung-Yi Lin Pengchong Jin Golnaz Ghiasi
Mingxing Tan Yin Cui Quoc V. Le Xiaodan Song
Google Research
{xianzhi,tsungyi,pengchong,golnazg,tanmingxing,yincui,qvl,xiaodansong}@google.com

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 that 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. SpineNet achieves state-of-the-art performance of one-stage object detector on COCO with 60% less computation, and outperforms ResNet-FPN counterparts by 6% AP. SpineNet architecture can transfer to classification tasks, achieving 6% top-1 accuracy improvement on a challenging iNaturalist fine-grained dataset.*

1. Introduction

In the past 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 the input image into intermediate features with monotonically decreased resolutions. Most improvements of network architecture design are in adding network depth and connections within each feature resolution group [19, 18, 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."*



	#FLOPs▲	AP
SpineNet-49s	**44.7B**	**40.6**
SpineNet-49	**92.8B**	**42.7**
R50-FPN [22]	96.8B	37
SpineNet-49 @896	**181.4B**	**45.6**
SpineNet-96	**283.8B**	**47.5**
R101-FPN [22]	325B	41.1
HRNetV2-W40 [32]	338.1B	41.6
SpineNet-143	**544.5B**	**49.2**
AmoebaNet-FPN [22]	655.5B	43.4
Mask R-CNN X152-FPN [8]	784.3	46.4
AmoebaNet-NAS-FPN [6]	1316.5B	48.3

Figure 1: **FLOPs (Multi-Adds) *vs*. AP** on COCO `test-dev`. Using the same RetinaNet detector framework, SpineNet outperforms FPN [22] and NAS-FPN [6] by a large margin.

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 de-

coder 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 2 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.

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 a large margin (*+3.7% AP*) in the object detection task. The result can be further improved (*+2.0% AP*) 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.

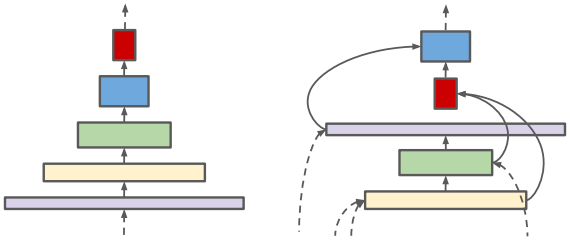


Figure 2: An example of scale-decreased network (left) *vs*. scale-permuted network (right). The width of block indicates feature resolution and the height indicates feature dimension. Dotted arrows represent connections from/to blocks not plotted.

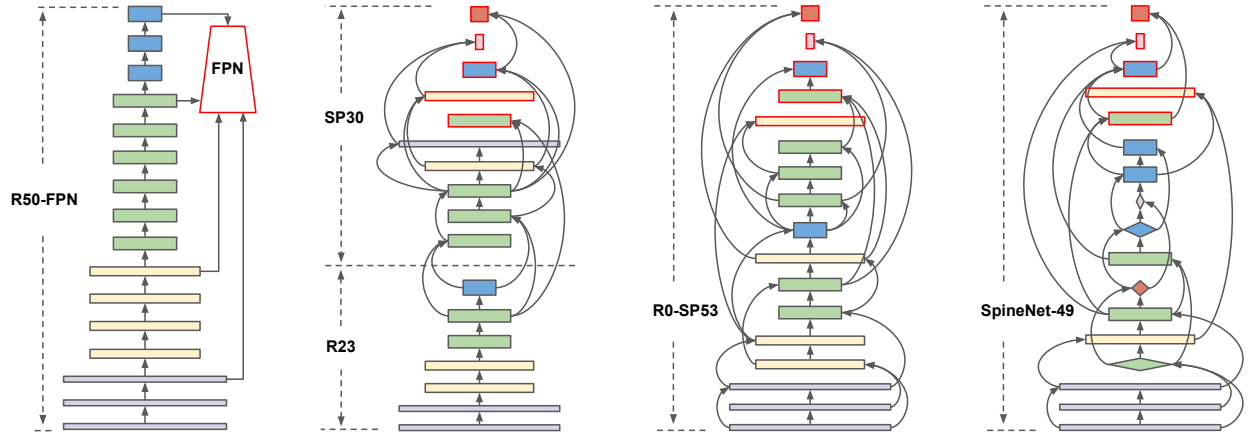
To this end, we show SpineNet achieves strong performance in object detection. But does it perform as well for image classification? To answer this question, we evaluate SpineNet on ImageNet and iNaturalist classification datasets. Even though SpineNet architecture is learned with object detection rather than classification tasks, it outperforms ResNet on both datasets. Particularly, SpineNet outperforms ResNet by 6% 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 to directly apply 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 connections [10, 35, 36, 34, 14, 13], enhancing model capacity [43, 17] and efficiency [3, 31, 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 [32] 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.



(a) R50-FPN: 37% AP @96.8B FLOPs.

(b) R23-SP30: 39.6% AP @96.5B FLOPs.

(c) R0-SP53: 40.7% AP @102.7B FLOPs.

(d) SpineNet-49: 42.7% AP @92.8B FLOPs.

Figure 3: **Building scale-permuted network by permuting ResNet.** All models are built with a similar computational budget in this figure. From (a) to (d), the computation gradually shifts from ResNet stem network to scale-permuted network. (a) The R50-FPN model, spending most computation in ResNet-50 followed by a FPN, achieves 37% 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 42.7% AP with learning additional block adjustments. Rectangle block represent bottleneck block and diamond block represent residual block. Output blocks are indicated by red border.

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, 28, 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.

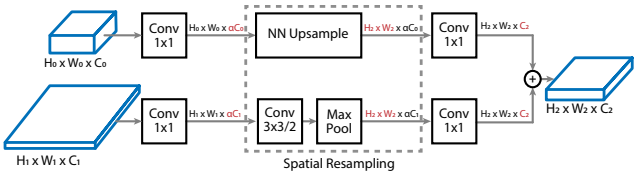


Figure 4: **Resampling operations.** Spatial resampling to upsample (top) and to downsample (bottom) input features followed by resampling in feature dimension before feature fusion.

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 4. 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 feature dimension C in a parent block to $\alpha \times C$. 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 $\alpha \times C$ to the target feature dimension. 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 3 shows a selection of learned model architectures in the family.

	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 3. 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 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 3d 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/49/96/143: 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 0.75. 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 repeats each block three times to grow the model depth and adjusts α in the resampling operation to 1.0. Figure 5 shows an example of increasing model depth by repeating blocks.

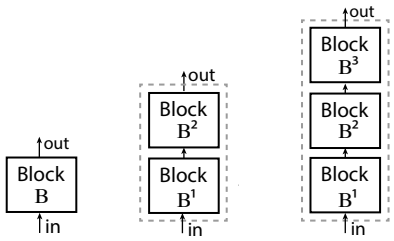


Figure 5: **Increase model depth by block repeat.** From left to right: blocks in SpineNet-49, SpineNet-96, and SpineNet-143.

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 may be orthogonal to this work. In Appendix A, we demonstrate an example of replacing building block type by MBConv [37] for mobile-size model with SpineNet.

4. Applications

4.1. Object Detection

The SpineNet architecture is learned with RetinaNet detector by simply replacing the default ResNet-FPN backbone model. We follow the architecture design for the class and box subnets in [22] for most models in SpineNet family. The only exception is that we reduce the feature dimension in subnets from 256 to 128 when using SpineNet-49s as the backbone model in order to balance the model capacity of backbone and subnets. We demonstrate that SpineNet can also be used as backbone model in Mask R-CNN detector [9] and improve both box detection and instance segmentation in a two-stage detector.

4.2. Image Classification

To demonstrate SpineNet has the potential to generalize to other visual recognition tasks, we use SpineNet for 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 [30] 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 deeper SpineNet. The long side of an image is resized to the target image size then the short side is padded with zeros to make a square image. Horizontal flipping and training scale jittering between $[0.4, 1.6]$ are implemented for data augmentation. For image classification, as in [7], we use standard scale and aspect ratio data augmentation. The network input is a 224×224 random crop from an augmented image or its horizontal flip. The input image is normalized by mean and standard deviation per color channel.

Training details: We adopt stochastic gradient descent (SGD) to train on Cloud TPU v3 devices with a weight decay of 4e-5 and a momentum of 0.9. For object detection, all detection models are *trained from scratch* on COCO `train2017` without ImageNet pre-training. We use a batch size of 256 to train for 250 epochs. The initial learning rate is set to 0.28 and a linear learning rate warmup is applied in the first 5 epochs. For RetinaNet models, we apply stepwise learning rate that decays to $0.1\times$, $0.01\times$, and $0.001\times$ at the last 40, 20, and 10 epoch. For Mask R-CNN models, we apply cosine learning rate decay. To obtain comparable results by training from scratch, we follow [8] to apply synchronized batch normalization with momentum 0.99 and implement DropBlock [5] with block size 3 and keep probability 0.9 for regularization. In both detection frameworks, we set anchor scale to 4, numbers of scales to 3, and aspect ratios to $\{1/2, 1, 2\}$. We follow common practices for other detector-specific settings [22, 9, 8]. For image classification, all models are trained with a batch size of 1024 for 90 epochs. We followed the same training strategy used in [7] with linear scaling of learning rate and gradual warmup in the first 5 epochs, except we use the cosine learning rate decay as suggested in [11]. Code will be publicly released for applying SpineNet to object detection and image classification.

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

backbone model	resolution	#FLOPs▲	#Params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SpineNet-49s	640×640	**44.7B**	**16.3M**	**40.6**	**59.3**	**43.7**	**20.9**	**44.6**	**55.1**
YOLOv3 DarkNet-53 [29]	608×608	70.3B	-	33.0	57.9	34.4	18.3	35.4	41.9
SpineNet-49	640×640	**92.8B**	**30.9M**	**42.7**	**61.6**	**45.8**	**23.1**	**46.7**	**57.2**
R50-FPN [22]	640×640	96.8B	34.0M	37.0	55.7	39.8	17.8	40.9	50.6
R50-NAS-FPN [6]	640×640	140.0B	60.3M	39.9	-	-	-	-	-
SpineNet-49	896×896	**181.4B**	**30.9M**	**45.6**	**64.9**	**49.3**	**27.7**	**48.6**	**58.3**
SpineNet-96	1024×1024	**283.8B**	**45.3M**	**47.8**	**67.3**	**51.8**	**30.3**	**50.8**	**59.6**
R101-FPN [22]	1024×1024	325.0B	53.0M	41.1	60.6	44.4	23.5	44.3	52.3
SpineNet-143	1280×1280	**544.5B**	**66.9M**	**49.2**	**68.9**	**53.4**	**32.4**	**52.5**	**60.3**
AmoebaNet-FPN [22]	1280×1280	655.5B	114.4M	43.4	-	-	-	-	-
R50-NAS-FPN[6]	1280×1280	1043.0B	103.0M	46.6	65.9	50.5	29.9	49.8	56.6
AmoebaNet-NAS-FPN [6]	1280×1280	1316.5B	166.5M	48.3	-	-	-	-	-

Table 2: **One-stage object detection results.** We compare using different backbones with RetinaNet except YOLOv3 [29]. Results are reported on single model without test-time augmentation. FLOPs of NAS-FPN and YOLOv3 are represented by Multi-Adds.

model	scale	type	#FLOPs	#Params	AP
R50-FPN	-	-	96.8B	34.0M	37.0
R35-SP18	-	-	91.7B	25.7M	38.6
R23-SP30	-	-	96.5B	28.7M	39.6
R14-SP39	-	-	99.7B	29.1M	39.4
R0-SP53	-	-	102.7B	29.5M	40.7
R0-SP53+	✓	-	92.0B	29.9M	42.1
SpineNet-49	✓	✓	92.8B	30.9M	42.7

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

model	resolution	AP	inference latency
SpineNet-49s	640×640	40.6	11.7ms
SpineNet-49	640×640	42.7	15.3ms
SpineNet-49	896×896	45.6	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).

the last 5 blocks built and allow output blocks to search from all existing blocks. At each sample, a proxy task is trained at 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 3, 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, the R0-SP53 model uses similar building blocks and gains 3.7% AP with a learned scale permutations and cross-scale connections. It is worth noting that the R14-SP39 model performs slightly worse than the R23-SP30 model, the hypothesis is using a small ResNet as the stem network throws away most spatial information at an early stage and it is challenging for following scale-permuted network to recover the spatial information. We further conduct ablation studies for the improvements made by additional block adjustments in SpineNet. The R0-SP53+ model adds scale adjustment in the search space, which gains 1.4% AP. The SpineNet-49 model further improves AP by 0.6% by adding both scale and block type adjustments.

detector	backbone model	resolution	#FLOPs	#Params	AP	AP_{50}	AP_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
Mask R-CNN(ours)	**SpineNet-49s**	640×640	**82.5B**	**19.2M**	**41.4**	**61.5**	**45.0**	**36.8**	**58.8**	**39.1**
Faster R-CNN	HRNetV2-W18[32]	800×1200	170.8B	26.2M	39.4	-	-	-	-	-
Mask R-CNN(ours)	**SpineNet-49**	640×640	**225.0B**	**43.6M**	**43.6**	**64.4**	**47.5**	**38.6**	**61.1**	**41.5**
Faster R-CNN	HRNetV2-W32[32]	800×1200	263.4B	45M	42.6	-	-	-	-	-
Mask R-CNN	R50-FPN [8]	800×1280	370.1B	45.8M	43.3	-	-	38.5	-	-
Mask R-CNN(ours)	**SpineNet-96**	1024×1024	**325.0B**	**57.5M**	**47.9**	**68.3**	**52.7**	**41.9**	**65.6**	**45.4**
Mask R-CNN	R101-FPN [8]	800×1280	445.9B	67.8M	45.0	65.7	49.3	39.5	62.5	42.1
Mask R-CNN(ours)	**SpineNet-143**	1280×1280	**518.8B**	**79.1M**	**49.3**	**69.8**	**54.2**	**43.2**	**67.2**	**47.0**
Mask R-CNN	X152-FPN [8]	800×1280	784.3B	142.6M	46.4	67.1	51.1	40.5	63.9	43.4

Table 5: **Two-stage object detection and instance segmentation results.** We measure the performance of SpineNets with our Mask R-CNN implementation using 1000 proposals. The detection performance of baseline Mask R-CNN is reported in [8] and its FLOPs and Params are reported in our Mask R-CNN implementation. The performance of HRNets [32] with Faster R-CNN are reported using 512 proposals in the open-sourced implementation. All results are on COCO `val2017` using single model without test-time augmentation.

network	ImageNet ILSVRC-2012 (1000-class)				iNaturalist-2017 (5089-class)			
	#FLOPs	#Params	Top-1 %	Top-5 %	#FLOPs	#Params▲	Top-1 %	Top-5 %
SpineNet-49s	**2.9B**	**14.2M**	**75.4**	**92.7**	**2.9B**	**15.2M**	**58.6**	**81.7**
ResNet-34	3.7B	21.8M	74.1	91.7	3.7B	23.9M	53.6	76.3
SpineNet-49	4.4B	**24.5M**	**77.2**	**93.5**	4.4B	**25.5M**	**60.6**	**83.2**
ResNet-50	**4.1B**	25.6M	76.4	93.1	**4.1B**	33.9M	54.2	76.6
SpineNet-96	**6.6B**	**38.9M**	**78.3**	**94.1**	**6.6B**	**40.0M**	**62.5**	**84.4**
ResNet-101	7.8B	44.6M	78.0	94.0	7.8B	52.9M	56.9	78.9
SpineNet-143	**9.7B**	60.5M	**79.0**	**94.5**	**9.7B**	**61.6M**	**64.4**	**85.7**
ResNet-152	11.5B	**60.2M**	78.7	94.2	11.5B	68.6M	58.3	80.0

Table 6: **Image classification results on ImageNet and iNaturalist.** Networks are sorted by increasing number of parameters on iNaturalist classification. 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.

5.4. Object Detection Results

RetinaNet: We evaluate SpineNet architectures on the COCO bounding box detection task with a RetinaNet detector. SpineNet consistently achieves 5%+ AP gain over the ResNet-FPN counterparts. The SpineNet-49, SpineNet-96, and SpineNet-143 backbones, using fewer FLOPs (Multi-Adds) than R50, R101, and AmoebaNet backbones, outperforms the counterparts by 5.7% AP, 6.4% AP, and 5.8% AP, respectively. In particular, SpineNet-143 achieves 49.2% AP on single model object detection without test-time augmentation. The results are summarized in Table 2.

Mask R-CNN: Here we show SpineNet achieves strong performance with Mask R-CNN detector in both box detection and instance segmentation. Table 5 shows the performance comparisons on `val2017` among SpineNets and other bakcbone models (*e.g.*, ResNet-FPN [8] and HRNet [32]) in Mask R-CNN framework. We measure the SpineNets performance in our Mask R-CNN implementation [1]. The HRNet performance is directly adopted from the open-sourced implementation [2]. Being consistent with RetinaNet results, SpineNets are able to use fewer FLOPs and parameters but achieve better AP and mask AP at various model sizes. Note that the SpineNet backbone architecture is learned on box detection with RetinaNet but works well with Mask R-CNN, showing the generality of SpineNet.

Real-time Object Detection: Our SpineNet-49s and SpineNet-49 with RetinaNet run at 30+ fps while achieving 40%+ AP 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. We report the average latency on 500 test images in Table 4.

[1] https://github.com/tensorflow/tpu/tree/master/models/official/detection

[2] https://github.com/HRNet/HRNet-Object-Detection

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) long-range connections; (2) short-range connections; (3) all connections then sequentially connecting every pair of adjacent blocks.

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.

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-Spine53, 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, the performance of SpineNet is on par with ResNet on ImageNet with similar FLOPs. On iNaturalist, SpineNet outperforms ResNet counterparts by a large margin of around 6%. 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 performance 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 64.5%/87.7% for SpineNet-49 and 59.2%/82.4% for ResNet-50, with a 5.3% improvement in Top-1 accuracy. The improvement of SpineNet-49 over ResNet-50 in Top-1 is 6.4% on the original iNaturalist dataset. Based on the experiment, we believe most of the improvement on iNaturalist is not due to capturing objects of variant scales but capturing subtle local visual differences thanks to the multi-scale features in SpineNet.

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-permuted models, we learn SpineNet by Neural Architecture Search in object detection and demonstrate it can be used directly in image classification. SpineNet achieves state-of-the-art object detection performance with only 40% model computation. The same SpineNet architecture achieves a better top-1 accuracy on ImageNet and 6% 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.

backbone model	#FLOPs	#Params	AP	AP_S	AP_M	AP_L
SpineNet-49xxs (MBConv)	**0.16B**	**0.85M**	**16.9**	**0.7**	**14.8**	**32.8**
MobileNetV3-Small-SSDLite [12]	0.16B	1.77M	16.1	-	-	-
SpineNet-49xs (MBConv)	**0.52B**	**1.01M**	**23.7**	**4.1**	**24.5**	**40.3**
MobileNetV3-SSDLite [12]	0.51B	3.22M	22.0	-	-	-
SpineNet-49s (MBConv)	**0.67B**	**1.34M**	**25.3**	**4.8**	**27.0**	**42.4**
MobileNetV2-NAS-FPNLite (3 @48) [6]	0.76B	2.16M	24.2	-	-	-
MobileNetV2-SSDLite [31]	0.8B	4.3M	22.1	-	-	-
MnasNet-A1-SSDLite [37]	0.8B	4.9M	23.0	3.8	21.7	42.0
SpineNet-49 (MBConv)	**0.97B**	**2.28M**	**27.3**	**5.0**	**29.8**	**45.4**
MobileNetV2-NAS-FPNLite (7 @64) [6]	0.98B	2.62M	25.7	-	-	-
MobileNetV2-FPNLite [31]	1.01B	2.2M	24.3	-	-	-

Table 9: **On-device object detection results.** Results are reported on single model without test-time augmentation on COCO `test-dev`.

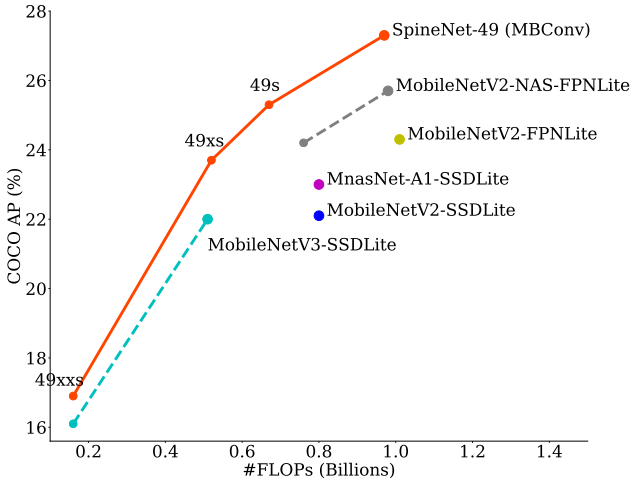


Figure 6: **FLOPs (Multi-Adds) *vs.* AP** on COCO `test-dev`. Built with MBConv blocks [37], SpineNet achieves state-of-the-art performance for on-device object detection.

model	#FLOPs	#Params	AP
SpineNet-49 (X-32 $\times$ 4)	91.8B	30.0M	42.2
SpineNet-49 (X-32 $\times$ 8)	111.9B	46.1M	43.1
SpineNet-49	92.8B	30.9M	42.7

Table 10: Results of building SpineNet-49 with ResNeXt blocks. Evaluated on COCO `val2017`.

Appendix A: On-Device Object Detection

We explore building SpineNet with MBConv blocks [37] for on-device object detection. Following [37], we set feature dimension $\{16, 24, 40, 80, 112, 112, 112\}$, expansion ratio 6, and kernel size 3×3 for MBConv blocks L_1 to L_7. 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-49xxs, SpineNet-49xs and SpineNet-49s by scaling the feature dimension of SpineNet-49 by $0.6\times$, $0.65\times$ and $0.75\times$ and setting the feature dimension in the box/class nets to 24, 40 and 48 respectively. We train all models in this section for 600 epochs at resolution 256×256 for SpineNet-49xxs and 384×384 for other models. DropBlock is removed during training. Except the above mentioned changes, all models are trained using the same settings as SpineNet-49 with the RetinaNet framework and evaluated on COCO `test-dev`. The results are presented in Table 9 and the FLOPs *vs.* AP curve is plotted in Figure 6.

Bulit with MBConv blocks, SpineNet-49xs/49s/49 use similar or less computation but outperform MnasNet, MobileNetV2, and MobileNetV3 by 2-3% AP. Note that as all the models in this section use handcrafted MBConv blocks from MnasNet, the performance should be no better than a joint search of SpineNet and MBConv blocks with NAS.

Appendix B: SpineNet with ResNeXt Blocks

We further explore building SpineNet with the ResNeXt blocks [40] by replacing the blocks in SpineNet-49 with either ResNeXt 32×4d or 32×8d blocks, following the practice in [40]. The resulting models are trained with the same settings as SpineNet-49 with RetinaNet and evaluted on COCO `val2017`. The results are given in Table 10.

network	feature level(s)	feature combination	feature dimension	learning rate schedule	ILSVRC-2012		iNaturalist-2017	
					Top-1 %	Top-5 %	Top-1 %	Top-5 %
ResNet-50	-	-	2048	step	76.1	92.9	54.4	76.9
SpineNet-49	P_3	-	256	step	74.2	92.1	56.4	79.4
SpineNet-49	P_4	-	256	step	75.3	92.7	58.3	81.0
SpineNet-49	P_5	-	256	step	75.9	92.9	58.8	81.5
SpineNet-49	P_6	-	256	step	75.4	92.7	59.8	82.2
SpineNet-49	P_7	-	256	step	73.8	91.8	56.2	79.5
SpineNet-49	P_3 - P_7	Concat	1280	step	76.6	93.3	56.8	79.3
SpineNet-49	P_3 - P_7	AvgFeaMap	256	step	76.7	93.3	59.6	82.6
ResNet-50	-	-	2048	cosine	76.4	93.1	54.2	76.6
SpineNet-49	**P_3 - P_7**	**AvgFeaMap**	**256**	**cosine**	**77.2**	**93.5**	**60.6**	**83.2**

Table 11: Additional image classification results on ImageNet and iNaturalist. Our final SpineNet classification model leverages features from all 5 levels (P_3 - P_7), combines them by upsampling and averaging their feature maps (AvgFeaMap) before global average pooling and is trained with cosine learning rate decay.

model	DropBlock	stochastic depth	AP
SpineNet-49	-	-	42.4
SpineNet-49	✓	-	42.7
SpineNet-49	-	✓	42.7
SpineNet-49	✓	✓	42.5

Table 12: Impact of different regularization methods for training SpineNet from scratch. Evaluated on COCO `val2017`.

Appendix C: Model Regularization

We use DropBlock [5] as the default regularization method to train SpineNet from scratch. It brings 0.3% AP gain without introducing extra cost at inference time. Besides DropBlock, stochastic depth [15] with drop connect ratio 0.2 can achieve a same performance improvement. However, combining DropBlock and stochastic depth leads to a 0.2% AP drop, which maybe due to excessive model regularization. Results are shown in Table 12.

Appendix D: Image Classification

We present SpineNet image classification results of using different levels of feature and feature combination strategies in Table 11. As can be seen from the results, even using the feature from a single level can give us reasonably good performance. For combining features from different levels, we also try to concatenate them after global average pooling of each level and yield a $256 \times 5 = 1280$ dimensional feature vector. Upsampling and averaging feature maps before global average pooling outperforms concatenation and is also much more compact. For learning rate strategy, training with cosine learning rate decay slightly outperforms step learning rate decay.

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. 1, 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. 2

[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. 5, 10

[6] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 1, 2, 3, 4, 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. 5

[8] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *ICCV*, 2019. 1, 5, 7

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 5

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 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. 5

[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. 2, 9

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 2

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1, 2

[15] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 10

[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. 2

[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. 2

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 1, 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. 2

[21] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 1, 2, 3, 4, 8

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 1, 2, 5, 6

[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. 2, 8

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

[29] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 6

[30] 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

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2, 9

[32] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, 2019. 1, 2, 7

[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. 2, 8

[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. 2

[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. 2

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2

[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. 5, 9

[38] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 2, 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] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 9

[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, 2

[44] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2, 5

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