Understanding the Effects of Pre-Training for Object Detectors via Eigenspectrum

Yosuke Shinya
DENSO CORPORATION
yosuke.shinya.j7r@jp.denso.com

Edgar Simo-Serra
Waseda University
ess@waseda.jp

Taiji Suzuki
The University of Tokyo / RIKEN
taiji@mist.i.u-tokyo.ac.jp

Abstract

ImageNet pre-training has been regarded as essential for training accurate object detectors for a long time. Recently, it has been shown that object detectors trained from randomly initialized weights can be on par with those fine-tuned from ImageNet pre-trained models. However, the effects of pre-training and the differences caused by pre-training are still not fully understood. In this paper, we analyze the eigenspectrum dynamics of the covariance matrix of each feature map in object detectors. Based on our analysis on ResNet-50, Faster R-CNN with FPN, and Mask R-CNN, we show that object detectors trained from ImageNet pre-trained models and those trained from scratch behave differently from each other even if both object detectors have similar accuracy. Furthermore, we propose a method for automatically determining the widths (the numbers of channels) of object detectors based on the eigenspectrum. We train Faster R-CNN with FPN from randomly initialized weights, and show that our method can reduce ~27% of the parameters of ResNet-50 without increasing Multiply-Accumulate operations and losing accuracy. Our results indicate that we should develop more appropriate methods for transferring knowledge from image classification to object detection (or other tasks).

1. Introduction

Object detection and instance segmentation are important tasks, with many real-world applications in robotics, healthcare, etc. Up until now, most detection and segmentation tasks have relied on ImageNet [59] fine-tuning [13, 82]. With fine-tuning, learned parameters or features of source tasks may be forgotten after learning target tasks [27], and domain similarity between tasks being important for transfer learning [84]. Furthermore, transferring knowledge between dissimilar tasks may cause negative transfer [58, 75]. Thus many works have discussed task difference between image classification and object detection [56, 65, 68, 6] and
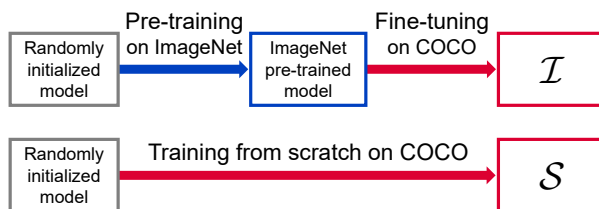
Do object detectors $\mathcal{I}$ and $\mathcal{S}$ converge to similar models?

Figure 1. R-CNN [13] shows that object detectors $\mathcal{I}$, which fine-tuned from ImageNet pre-trained models, can achieve high accuracy. "Rethinking ImageNet Pre-training" [18] shows that object detectors $\mathcal{S}$, which trained from scratch, can achieve similar accuracy to $\mathcal{I}$ under appropriate conditions. In this paper, we show that $\mathcal{I}$ and $\mathcal{S}$ behave differently from each other even if both object detectors have similar accuracy.

the effects of pre-training for object detectors [25, 69, 48, 31]. However, the influence caused by the task difference is still an open problem, and what and how to transfer knowledge from image classification to object detection are unclear. Avoiding these problems, it was recently shown that models trained on COCO [36] from random initialization can be on par with models trained (fine-tuned) from ImageNet pre-trained models [18], but it is not clear whether or not models with similar performance have different properties. To further understand the effects of fine-tuning object detectors, we analyze the eigenspectrum dynamics of the covariance matrix of each feature map in object detectors, and propose a method to automatically determine the numbers of channels necessary for performance. (Each feature map includes channel dimension in this paper.)

More specifically, motivated by the accurate object detectors trained from scratch [18, 86], we focus on the following research question. *Do object detectors fine-tuned from ImageNet pre-trained models and those trained from scratch converge to similar models?* If the answer is "Yes," we will have a better understanding of the task difference and the behavior of deep neural networks, and if the answer is "No, these object detectors do not converge to similar

models, but show similar accuracy by chance," we should incorporate the benefits of both object detectors.

To answer this question, we train object detectors as shown in Figure 1, and analyze the redundancy of feature maps in the detectors. To be more precise, we analyze the intrinsic dimensionalities of the feature maps, which represent how much information the feature maps memorize. Intrinsic dimensionalities can be quantified by calculating the eigenspectra of the covariance matrices of the feature maps, and are related to generalization error [70]. In this paper, we use the numbers of eigenvalues greater than a threshold as a simple metric of intrinsic dimensionalities, and we call the sets of intrinsic dimensionalities in a certain network the *intrinsic architecture*.

Our contributions are as follows.

- We analyze the eigenspectrum dynamics of the covariance matrix of each feature map in object detectors, and show that object detectors trained from ImageNet pre-trained models and those trained from scratch behave differently from each other even if both object detectors have similar accuracy.

- We propose a method for automatically determining the widths (the numbers of channels) of object detectors. We report the results of Faster R-CNN with FPN trained from scratch, and show that our method can reduce ∼27% of the parameters of ResNet-50 without increasing Multiply-Accumulate operations (MACs) and losing accuracy, and can improve COCO AP by 0.3% without increasing parameters (See Sec. 4.5).

- We explain why architectures and learning schedules of prior object detectors trained from scratch work well in Sec. 5.2. We bridge recent theoretical analysis on generalization error [70] and the experimental results of recent object detectors [18].

2. Related Work

2.1. Neural Network Generalization

One of the most important mysteries of neural networks is its generalization ability. To understand it, some work has discussed the relation between generalization and compressibility [66, 50, 1, 70]. Information Bottleneck [66, 62] and Canonical Correlation Analysis (CCA) [53, 50] are used for analyzing the dynamics of neural networks. [66] showed that training with Stochastic Gradient Descent (SGD) has two phases (a label fitting phase and a representation compression phase). [62] shows that networks with ReLU do not necessarily exhibit the compression phase, and that fitting to task-relevant information and the compression of task-irrelevant information occur simultaneously. [50] shows that generalizing/larger networks converge to more similar solutions than memorizing/smaller networks. Using

CCA, Transfusion [54] analyzes the effects of pre-training for classifying medical images, which are clearly different from natural images in ImageNet and COCO.

The most related theoretical analysis to this paper is the *degree of freedom* of reproducing kernel Hilbert spaces (RKHSs), which is defined in [70]. Suzuki [70] shows the following two important properties of neural networks which motivate our work. (i) "if the eigenvalues of the kernels decreases rapidly, then the degree of freedom gets smaller, and we achieve a better generalization by using a simpler model." (ii) "the effective dimension of the network is less than the actual number of parameters." Spectral-Pruning [71], which uses the degree of freedom [70] as the intrinsic dimensionality of models, is applicable to compress complicated networks. Our method and analysis are based on eigenspectrum [70, 71] and the dynamics of neural networks [66, 53, 50]. However, these prior works do not analyze the behavior of neural networks when fine-tuned for object detection from ImageNet pre-trained models.

2.2. Neural Architecture Search (NAS)

NAS has been a hot research topic on deep learning since the success of NAS with reinforcement learning [87], and efficient methods have broadened its applicability [88, 38, 2]. Genetic CNN [79] and NASNet [88] transfer architectures learned on a proxy dataset (*e.g.*, CIFAR-10) to large-scale datasets (*e.g.*, ImageNet). On the other hand, ProxylessNAS [2] reduces memory consumption by path-level binarization, and directly learns architectures for a large-scale dataset. In addition to NAS for image classification, a few works focus NAS for semantic segmentation [63, 4, 37] and object detection [11, 5]. NAS-FPN [11] and DetNAS [5] search the architectures of Feature Pyramid Networks [35] and backbones for object detectors respectively. However, these prior works [11, 5] do not determine the widths of feature maps automatically, and computational costs for training will become higher if their search space includes the widths.

Determining the widths of feature maps in CNNs can be considered as a subset of NAS. Although various approaches have been proposed [9, 21, 8, 43], shrink-and-expand [15, 52] is a more suitable approach for object detectors because of its simplicity and scalability. MorphNet [15] shrinks and linearly expands networks. The shrinking imposes L1 regularization on the scaling factors of Batch Normalization to identify and prune unimportant channels like Network Slimming [42], and takes into account specific resource constraints (*e.g.*, the number of floating point operations). Neural Rejuvenation [52] revives dead neurons (reallocates and reinitializes useless channels) during training. Although the effectiveness of these methods [15, 52] is verified on ImageNet, it is unknown whether these methods can be applied to object detectors.
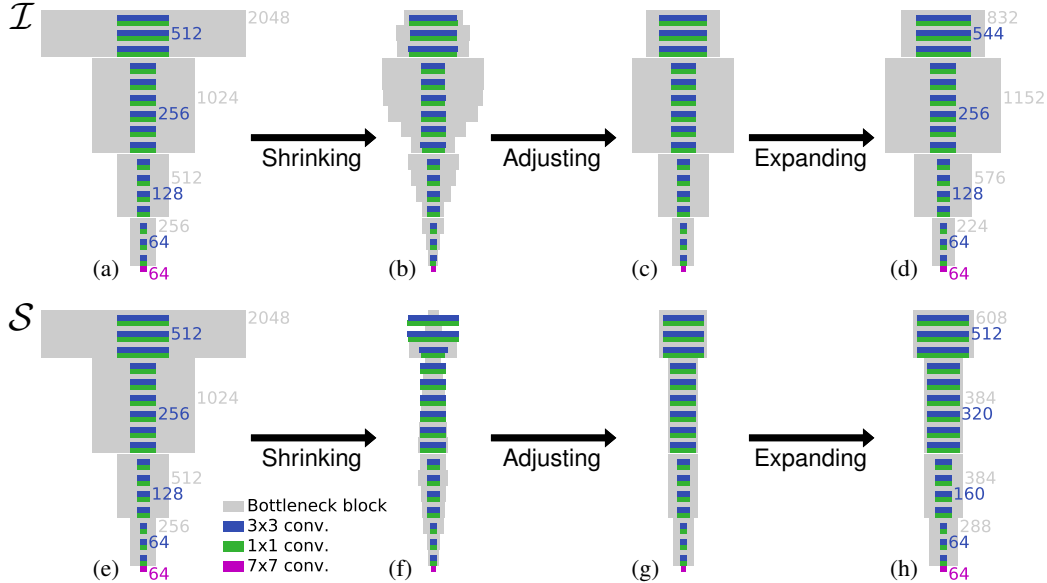
Figure 2. Overview of Intrinsic Architecture Search. The width of each rectangle is proportional to the output width of each layer. 7×7, 1×1, and 3×3 convolutions are denoted in purple, green, and blue respectively, and bottleneck building blocks are denoted in gray like [15]. (a): The ResNet-50 backbone of an object detector $\mathcal{I}$ fine-tuned from an ImageNet pre-trained model. (e): The ResNet-50 backbone of an object detector $\mathcal{S}$ trained from scratch. (a) and (e) have the same (extrinsic) architecture, and $\mathcal{I}$ and $\mathcal{S}$ have similar accuracy. Thus the difference between $\mathcal{I}$ and $\mathcal{S}$ is unclear. To clarify it, we shrink (by extracting intrinsic architectures), adjust, and expand widths. See Sec. 3.2 for the details of our algorithm. See Sec. 4.3 for the details of models used in our experiments.

2.3. Object Detection and Instance Segmentation

Object detection is one of the core technologies in computer vision, and has advanced rapidly with deep neural networks [64, 10, 13, 12, 57, 55, 41, 35, 68] (Refer to a survey [39] for details). In addition, instance segmentation [7, 33, 19, 40], which is the task of segmenting and classifying individual objects, is important for further detailed object recognition. Most methods for these tasks train models from ImageNet pre-trained models for better accuracy. However, pre-training backbones in object detectors on image classification dataset causes learning bias and limits architecture design [65, 86].

To avoid the problems of pre-training, training object detectors from scratch (from randomly initialized weights) has been discussed in some literature [65, 32, 34, 28, 86, 18]. DSOD [65] shows that deep supervision [29] is critical for training single-shot object detectors from scratch, and adopts implicit deep supervision via dense connections [24]. ScratchDet [86] shows that Batch Normalization [26, 61] helps training from scratch to converge, and redesigns the backbone of single-shot object detectors. [18] shows that Mask R-CNN trained from scratch with appropriate normalization and longer training (instead of pre-training) can be on par with those fine-tuned from ImageNet pre-trained models.

The most similar work to ours is DetNet [34], which is a specialized backbone for object detection. DetNet mainly focuses on scales (the receptive fields and the spatial resolutions of feature maps) to overcome drawbacks of ImageNet pre-trained models designed for image classification. However, its widths are manually determined. On the other hand, our method does not aim to determine the spatial resolutions. Using our method and DetNet complementarily would be beneficial.

3. Intrinsic Architecture Search

In this section, we propose a method for automatically determining the widths (the numbers of channels) of feature maps. Our method reflects intrinsic architectures by calculating the redundancy of feature maps, and is applicable to complicated networks, such as Faster R-CNN with FPN and Mask R-CNN. Figure 2 shows an overview of our method. We call our algorithm *Intrinsic Architecture Search*, and we call architectures discovered by our algorithm *ResiaNet* whose base backbone is ResNet.

3.1. Determining Widths

Optimizing the widths of feature maps is formulated as

$$O_{1:M}^* = \operatorname*{argmin}_{c(O_{1:M}) \leq \zeta} \min_{\theta} \mathcal{L}(\theta), \qquad (1)$$

where M is the total number of layers, $O_{1:M}$ are the widths of output feature maps, θ is the parameters (weights) in neural networks, $\mathcal{L}$ is a loss function for training neural net-

works, c is a function for calculating resource consumption (*e.g.*, Multiply-Accumulate operations (MACs)), and ζ is a specified maximum allowable resource consumption. This formulation is exactly the same as [15], and most notations in this section and some descriptions in Algorithm 1 follow MorphNet [15] for ease of comparing methods.

Although MorphNet [15] and Neural Rejuvenation [52] also tackle the determination of widths, these methods need to change training and intrinsic dimensionalities. In addition, applying them to object detection and instance segmentation poses some difficulties below. (i) These methods depend on Batch Normalization [26]. Therefore, applying them to networks with other normalization layers [78, 46] is not trivial. Furthermore, when we apply them to networks without normalization layers [85], we need to add Batch Normalization layers [52]. (ii) These methods use additional regularizers. Since object detection and instance segmentation are multi-task learning including classification and localization, we might need to balance regularization. (iii) These methods need to train multiple models [15] or tune additional hyperparameters [52]. This is a serious problem especially for object detection and instance segmentation because training for these tasks takes a long time (See model zoos of [14, 3, 49]).

3.2. Overview

We propose a method for determining the widths of object detectors using eigenspectrum [70]. Algorithm 1 shows the whole process, where $S_{1:M}$ are the eigenspectra of feature maps, $d_{1:M}$ are the intrinsic dimensionalities of the feature maps, T is a threshold for calculating intrinsic dimensionalities (*e.g.*, 10^{-3}), len is a function for counting numbers which meet the condition, and ω is a width multiplier.

The details of Algorithm 1 are described below. In **Step 1**, we set initial weights. Weights in a base backbone (*e.g.*, ResNet-50) are initialized from one of the ImageNet pretrained models, or randomly initialized. Weights out of the base backbone are randomly initialized. In **Step 2**, we train the whole network (*e.g.*, Faster R-CNN with FPN or Mask R-CNN) with the base backbone. In **Step 3**, we calculate the eigenspectrum of each feature map in the whole network (See Sec. 3.3 for details). Eigenvalues are normalized with the largest eigenvalue of each feature map. In **Step 4**, we shrink the widths of each feature map by extracting an intrinsic architecture (See Sec. 3.4 for details). In **Step 5**, we adjust the widths mainly for networks with multiple branches (See Sec. 3.5 for details). In **Step 6**, we expand the widths by linear expanding (See Sec. 3.6 for details).

3.3. Calculating Eigenspectra

When we calculate the eigenspectra of feature maps which have spatial resolutions (*i.e.* almost all feature maps in CNNs), we normalize the covariance matrices by the res-

---

Algorithm 1 Intrinsic Architecture Search
1: Set initial weights.
2: Train the whole network to find
   $\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta)$.
3: Calculate eigenspectra $S_{1:M}$.
4: Calculate intrinsic dimensionalities $d_{1:M}$ by
   $d_{1:M} = \operatorname{len}(S_{1:M} > T)$.
5: Determine new widths $O'_{1:M}$ by adjusting $d_{1:M}$.
6: Find the largest ω such that $c(\omega \cdot O'_{1:M}) \leq \zeta$.
7: **return** $\omega \cdot O'_{1:M}$.

---

olutions. Specifically, the (non-centered) covariance matrix Σ of a feature map F is calculated as

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{W_i H_i} \sum_{x=1}^{W_i} \sum_{y=1}^{H_i} F_{i,x,y} F_{i,x,y}^{\top}, \qquad (2)$$

where n is the number of images (we randomly sample 5,000 images from the training set in our experiments), W_i, H_i are the spatial width and height of the feature map, and $F_{i,x,y}$ is a feature vector whose coordinates are (x, y) in the feature map for the i-th image. Not only $F_{i,x,y}$ but also W_i, H_i depend on images feed-forwarded because input image resolutions may change in the case of COCO.

We calculate the eigenspectra of feature maps before or after convolutional layers, fully connected layers, and transposed convolutional layers. Note that feature maps after L-th convolutional layer and feature maps before $(L + 1)$-th convolutional layer generally do not match due to normalization layers and activation layers.

3.4. Shrinking Widths

We calculate intrinsic dimensionalities from the eigenspectra. We use the numbers of eigenvalues greater than a predefined threshold as intrinsic dimensionalities. (Using degree of freedom [71] may be better, though we do not use pruning and we set random values to the initial weights.) Although we set 10^{-3} to the threshold in our experiment, we may get better accuracy if we tuned the threshold as a hyperparameter.

3.5. Adjusting Widths

If the network has multiple branches, adjusting intrinsic dimensionalities is necessary to determine new widths, because either the input feature maps or the output feature maps of branches may have to have the same widths. Especially for ResNet with bottlenecks, where the widths of feature maps which pass through shortcuts are set to the maximum intrinsic dimensionalities in the same stage for preserving most information which flows shortcut. Furthermore, we set the same output widths to the first and the
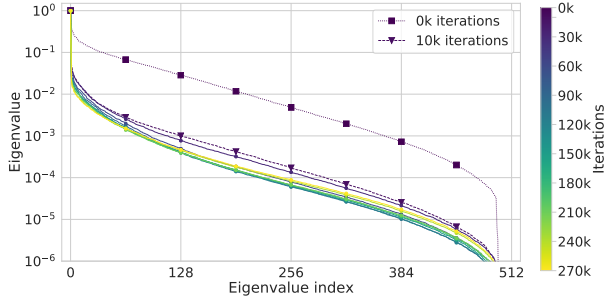
Figure 3. Dropping of eigenvalues. Some feature maps in the stage 5 of ResNet-50 become highly redundant in the first 10k iterations, though it is irredundant before fine-tuning (0k iterations).

Figure 4. Rebounding of eigenvalues. The numbers of eigenvalues greater than 10^{-3} increase immediately after the first learning rate decay (210k iterations) in some feature maps.

second convolutional layers of all residual blocks in the same stage by calculating the geometric mean of intrinsic dimensionalities. This setting has some advantages: (i) The second convolutional layers of residual blocks can be replaced with depthwise convolutional layers like [60]. (ii) Using the same widths is efficient considering memory access cost [47]. (iii) Implementation is easy and thus modifications to the code of ResNet are minimized.

3.6. Expanding Widths

Our expanding method is basically the same as that of [15]. Specifically, the output width of each layer is multiplied by a uniform width multiplier ω to fit a target resource consumption. The optimal ω can be found by a binary search because $c(\omega \cdot O'_{1:M})$ monotonically increases with ω in our experiments. $c(O_{1:M})$ is calculated as

$$c(O_{1:M}) = \sum_{L=1}^{M+1} I_L O_L K_L^2 W_L H_L, \qquad (3)$$

when targeting MACs, and

$$c(O_{1:M}) = \sum_{L=1}^{M+1} I_L O_L K_L^2, \qquad (4)$$

when targeting the number of parameters, where I_L, O_L are the widths of the input/output feature map, K_L is the kernel size, and W_L, H_L are the spatial width and height of the output feature map, for each layer $L = 1, \ldots, M+1$. We consider O_{M+1} is a fixed number (*e.g.*, 1,000 for ImageNet classification). For simplicity, we consider the spatial width and height of kernel size to be the same in each layer, and omit the resource consumption of biases.

To avoid odd widths [15], we round $\omega \cdot O'_{1:M}$ to hardware-friendly multiples (*e.g.*, multiples of 4, 8, 16, or 32). This rounding is also useful for networks with Group Normalization layers [78]. When resource fittings are too coarse, we may fill the gaps by increasing widths greedily.
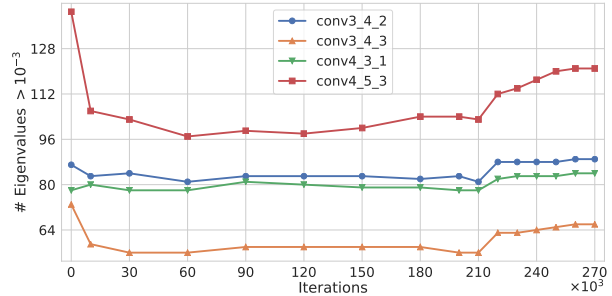
4. Experiments

To analyze the effects of pre-training for object detectors and to verify the effectiveness of our method, we conduct experiments on COCO.

4.1. Experimental Settings

The experimental settings mainly follow Mask R-CNN [19] in Detectron [14] (which includes implementation by the authors of Mask R-CNN) like [18]. Our implementation is based on Detectron.pytorch [74], which is a PyTorch implementation of Detectron.

We use ResNet-50 [20] as a base backbone. We train Faster R-CNN [57] with Feature Pyramid Network (FPN) [35] and Mask R-CNN [19] in an end-to-end manner [57]. We use Group Normalization (GN) [78], because appropriate normalization is a key factor for training from scratch [86, 18], and GN has several advantages [78] compared to Synchronized Batch Normalization [51]. The learning rate settings follow [18]. Specifically, the initial learning rate is 0.02 with warm-up [16], and the learning rate is reduced by $10\times$. Iterations for the first decay, the second decay, and ending training are 60k, 80k, 90k for $1\times$ schedule, 120k, 160k, 180k for $2\times$ schedule, and 210k, 250k, 270k for $3\times$ schedule. We use synchronous SGD with an effective batch size of 16 (= 2 images/GPU $\times$ 8 GPUs), a momentum of 0.9, and a weight decay of 10^{-4}.

All models are trained on COCO `train2017` set (118,287 images) and evaluated on COCO `val2017` set (5,000 images) with COCO metrics unless otherwise stated.

4.2. Eigenspectrum Dynamics

To analyze the effects of pre-training for object detectors, we observed the dynamics of the eigenspectrum of Mask R-CNN. Figure 3 shows the eigenspectrum of a feature map after the conv5_1_3 (the third convolutional layer in conv5_1 bottleneck building block. We call the convolutional layers of ResNet in this manner.) of ResNet-50. In the case of this layer, the eigenspectrum drops fast in the
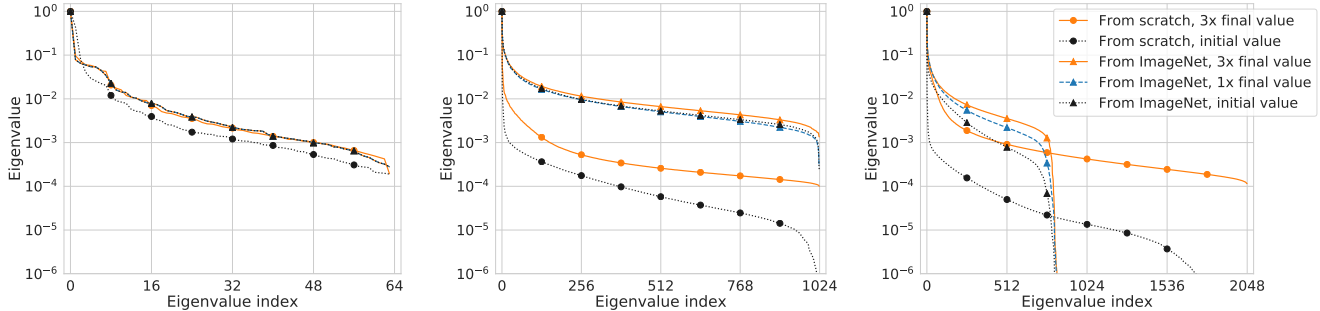
Figure 5. Intrinsic dimensionalities. Object detectors trained from ImageNet pre-trained models *vs.* those trained from scratch. These detectors behave differently from each other even if both detectors have similar accuracy. **Left**: Feature maps before conv2_1_1. Lower layers in backbones converge to similar eigenspectra. Freezing the weights initialized from ImageNet pre-trained models in lower layers is a reasonable choice. **Middle**: Feature maps before conv5_1_1. Higher layers in backbones converge to dissimilar eigenspectra. **Right**: Feature maps before conv5_2_1. In the case of the models trained from an ImageNet pre-trained model with GN layers, the feature maps close to the output layer (for ImageNet classification) do not use over half of the dimensions, and they are not reused even after fine-tuning.

first 10k iterations. Similar behavior can be seen in feature maps with $32\times$ strides after conv5_2_3, conv5_3_3, and projection shortcut in conv5_1.

This result demonstrates that some information obtained in ImageNet pre-training is forgotten. There are three possible reasons. (i) Features for 1000-class image classification are too rich for most object detection tasks (*e.g.*, Classification ability needed for COCO detection is 81-class classification including a background class). (ii) In pre-training on ImageNet, the stage 5 of ResNet is very close to the output layer. Layers which are close to the output layer may compress information to minimum needed for the pre-training task. (iii) The strides of conv5_x are too coarse to localize objects. DetNet [34] and ScratchDet [86] also discuss this problem and change the strides for object detection. Unlike these works, our finding is that SGD (with other regularization methods) automatically limits the intrinsic dimensionalities of standard ResNet without changing the strides.

Eigenspectrum dynamics can capture not only the forgetting of ImageNet pre-trained features but also the acquisition of features for COCO. Figure 4 shows the numbers of eigenvalues greater than 10^{-3}. Eigenvalues first down, then up, in some layers. This rebound occurs when the learning rate decays and may relate to the learning rate schedules and a finding in [18] (See discussions in Sec. 5.2).

4.3. Intrinsic Architecture

Here, we investigate whether models fine-tuned from an ImageNet pre-trained model and a model trained from scratch converge to similar intrinsic architectures.

We compare three models below. (i) $\mathcal{S}3$ trained from scratch with $3\times$ schedule (AP^{bbox}: 39.0%, AP^{mask}: 34.8%), (ii) $\mathcal{I}1$ trained from an ImageNet pre-trained model with $1\times$ schedule (AP^{bbox}: 38.9%, AP^{mask}: 34.6%), and (iii) $\mathcal{I}3$ trained from an ImageNet pre-trained model with $3\times$ schedule (AP^{bbox}: 40.3%, AP^{mask}: 35.7%).

Figure 2(b) and Figure 2(f) show the intrinsic architectures of $\mathcal{I}1$ and $\mathcal{S}3$, and Figure 5 shows some characteristic intrinsic dimensionalities. The intrinsic architecture of the model trained from scratch ($\mathcal{S}3$) is different from that of the models trained from the ImageNet pre-trained model ($\mathcal{I}1$, $\mathcal{I}3$), even if the models show similar AP ($\mathcal{S}3$ *vs.* $\mathcal{I}1$). The accuracy of object detectors will be improved if we properly incorporate the benefits of ImageNet pre-training and random initialization.

4.4. Discovered Backbones

Next, we apply Intrinsic Architecture Search to $\mathcal{S}3$, $\mathcal{I}1$, and $\mathcal{I}3$ for new backbones. Figure 2(d) and Figure 2(h) show the architectures of ResiaNet $\mathcal{I}1$-50 (MACs) and ResiaNet $\mathcal{S}3$-50 (MACs), whose target resource consumption is the MACs of ResNet-50. The architecture of ResiaNet $\mathcal{I}3$-50 is similar to that of ResiaNet $\mathcal{I}1$-50. Specifically, its width settings (the numbers in Figure 2(d) from below) are (64, 64, 224, 128, 576, 256, 1152, 544, 896) for ResiaNet $\mathcal{I}3$-50 (MACs), and (64, 64, 256, 160, 608, 288, 1216, 544, 960) for ResiaNet $\mathcal{I}3$-50 (params) whose target is the number of parameters of ResNet-50.

ResiaNet $\mathcal{I}1$-50 and ResiaNet $\mathcal{I}3$-50 have fewer widths in stage 5 and have more widths in stages 3 and 4 than ResNet. Reducing widths in stage 5 is caused by the characteristic of models trained from ImageNet pre-trained models (Figure 5 Right). Increasing widths in stages 3 and 4 may be caused by the object scales in COCO and the number of residual blocks (The information which flows through shortcuts is stacked gradually (Figure 2(b)), and the total amount of information may depend on the number of residual blocks). By contrast, ResiaNet $\mathcal{S}3$-50 does not widen the widths of feature maps which pass through shortcuts in stages 3 and 4 so much. We conjecture that this flat architecture is effective for maintaining edge information and localizing objects, but not suitable for classification.

Backbone	Normalization	Classification		COCO (2× schedule)						COCO (1×)
		MACs	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP
ResNet-50 [34]	SyncBN	3.8 G	—	34.5	55.2	37.7	20.4	36.7	44.5	—
ResNet-50*	GN	4.09 G	25.5 M	35.5	55.6	38.5	21.3	37.5	45.3	29.4
ResiaNet $\mathcal{S}$3-50 (MACs)	GN	4.06 G	18.6 M	35.4	55.4	38.6	21.5	37.3	45.2	28.9
ResiaNet $\mathcal{I}$1-50 (MACs)	GN	4.05 G	21.7 M	35.5	55.5	38.6	21.4	37.3	**46.0**	29.2
ResiaNet $\mathcal{I}$3-50 (MACs)	GN	4.07 G	22.0 M	35.4	55.6	38.4	21.3	37.8	45.5	29.3
ResiaNet $\mathcal{I}$3-50 (params)*	GN	4.92 G	24.7 M	**35.8**	**55.9**	**38.9**	**21.8**	**38.0**	45.6	—
DetNet-59 [34]	SyncBN	4.8+ G	—	**36.3**	**56.5**	**39.3**	22.0	**38.4**	**46.9**	—
DetNet-59†	GN	5.00+ G	18.3+ M	36.2	56.0	**39.3**	22.1	38.3	46.0	—
DetiaNet $\mathcal{I}$2-59 (MACs)	GN	4.94+ G	17.4+ M	36.2	56.0	**39.3**	**22.5**	38.1	46.0	—

Table 1. Efficiency on COCO object detection. All detectors are trained from randomly initialized weights. The characters like $\mathcal{S}$3 and $\mathcal{I}$1 in backbone names denote which models are used to determine widths. SyncBN: Synchronized Batch Normalization, GN: Group Normalization. +: Additional MACs and #params for an additional stage out of the backbone are needed. †: Our implementation. ⋆: We show the mean of five runs in the columns of COCO (2× schedule). The difference of COCO AP between these two backbones is statistically significant ($p < 0.05$ in two-sided Welch's t-test).

4.5. Efficiency on COCO Object Detection

To quantify the impact on accuracy caused by the difference of intrinsic architectures and identify better backbones than ResNet, we trained Faster R-CNN with FPN from scratch. Table 1 shows the results.

ResiaNet $\mathcal{S}$3 (MACs), ResiaNet $\mathcal{I}$1 (MACs), and ResiaNet $\mathcal{I}$3 (MACs), which trained with 2× schedule, achieve similar AP to ResNet with fewer parameters than ResNet. ResiaNet $\mathcal{S}$3 (MACs) has ∼27% fewer parameters than ResNet, and it is the most efficient. However, it may slightly degrade classification accuracy considering AP_{50}. ResiaNet $\mathcal{I}$3 (params) achieves better AP than ResNet with the similar number of parameters. (Note that simple width multipliers [23, 83] cannot improve AP without increasing parameters. In addition, they degrade AP by ∼0.6% to reduce parameters by ∼27%.)

ResiaNet $\mathcal{I}$1 (MACs) and ResiaNet $\mathcal{I}$3 (MACs) achieve higher AP than ResiaNet $\mathcal{S}$3 (MACs) if they are trained with 1× schedule. Thus, the intrinsic architectures of $\mathcal{I}$1 and $\mathcal{I}$3 have the effect of speeding up convergence. These results are different from [18] because we reinitialize weights. Besides, the differences of AP by the schedules indicate that using shorter training as a proxy task [11] is insufficient for this case.

In addition, we verify the effectiveness of DetiaNet $\mathcal{I}$2-59 (MACs), whose base backbone is DetNet-59 with GN (AP^{bbox}: 39.9%) which trained from an ImageNet pretrained model with 2× schedule. We set $10^{-3.5}$ to the threshold for eigenvalues because the number of parameters increases if it is 10^{-3}. Table 1 shows the results. DetiaNet $\mathcal{I}$2 (MACs) achieves similar AP to DetNet with ∼5% fewer parameters than DetNet. Although the parameter reduction of DetNet is more difficult than that of ResNet, our method is also effective for DetNet.

Backbone	#params	Top-1 err	Top-5 err
ResNet-50*	25.5 M	23.78	6.97
ResiaNet $\mathcal{S}$3-50 (MACs)	18.6 M	24.71	7.40
ResiaNet $\mathcal{I}$1-50 (MACs)	21.7 M	24.04	7.18
ResiaNet $\mathcal{I}$3-50 (MACs)*	22.0 M	23.83	6.98
ResiaNet $\mathcal{I}$3-50 (params)*	24.7 M	**23.45***	**6.85***

Table 2. Evaluation with transferring from COCO to ImageNet. ⋆: We show the mean of five runs. *: Statistically significant differences from ResNet-50 ($p < 0.05$ in two-sided Welch's t-test).

4.6. Efficiency on COCO Instance Segmentation

To verify effectiveness on instance segmentation, we also trained Mask R-CNN from scratch with 2× schedule. ResiaNet $\mathcal{I}$3-50 (MACs) achieves similar AP (AP^{bbox}, AP^{mask}: 36.6%, 33.1%) to ResNet-50 (36.6%, 33.0%). ResiaNet $\mathcal{S}$3-50 (MACs) has slightly lower AP (36.5%, 32.8%). This result means that the parameter reduction of Mask R-CNN is more difficult than that of Faster R-CNN, and reflects that the intrinsic dimensionalities of networks trained on difficult tasks are large [71].

4.7. Transferring Architecture to ImageNet

We investigate whether ResiaNet also improves parameter efficiency if we transfer the intrinsic architectures of the models trained on COCO to ImageNet classification. Table 2 shows the results. ResiaNet $\mathcal{S}$3 (MACs) has higher error rates than ResNet. Its widths are effective for COCO but not suitable for ImageNet classification. ResiaNet $\mathcal{I}$3 (MACs) achieves similar error rates to ResNet with fewer parameters than ResNet. This result indicates that the widths of ResiaNet $\mathcal{I}$3 mainly depend on the redundancy inherited from an ImageNet pre-trained model (Figure 5 Right).

5. Discussion and Conclusions

In this section, we first summarize our results and discuss the need to develop appropriate knowledge-transfer methods for object detectors. After that, we discuss why architectures and learning schedules of prior work, which trains object detectors from scratch, work well. Finally, we describe the limitations and weakness of our method.

5.1. Appropriate Knowledge Transfer

Although ImageNet pre-training increases intrinsic dimensionalities in higher layers (Figure 2(b)), the increase of parameters caused by them does not improve COCO AP (Table 1). These results do not necessarily mean that ImageNet pre-training is inefficient and meaningless for object detection. This is because the increase of parameters in higher layers brings us better classification ability (Table 2). The problem is not ImageNet pre-training itself but rather the forgetting of ImageNet pre-trained features (Figure 3). We need to take care of the compression of task-irrelevant information [62]. Information for classification may be regarded as task-irrelevant for localization, and vice versa.

Considering the above-mentioned results, the current standard architectures and fine-tuning methods of object detectors are insufficient for utilizing pre-training. For training better object detectors, methods for appropriately transferring the knowledge of ImageNet will be needed. The ideas of Decoupled Classification Refinement (DCR) [6] will be helpful. [6] decouples features for classification and localization, and the added classifier is trained not to forget translation-invariant ImageNet pre-trained features. To improve the efficiency of DCR, multi-task learning with automatic branching [44] may also be needed.

5.2. Understanding Prior Work with Our Results

DetNet [34] and ScratchDet [86] eliminate feature maps with $32\times$ strides from backbones, and weigh those with finer strides relatively heavily. These manual designs can imitate the architecture in Figure 2(h). Considering the feature forgetting (Sec. 4.2), the designs can avoid wasting parameters even if detectors are pre-trained. Choosing strides automatically with [63, 37, 76, 2] will be more effective.

DetNet [34] uses 1×1 convolution projection instead of identity mapping although stages 4, 5, and 6 have the same spatial resolution. Our results (Figure 5 Right) imply that the design keeps stages 4 and 5 away from the output layer, and avoids too sparse representation.

Our results (Figure 5 Right) also imply that current pre-training for object detectors can be considered as deep supervision [29]. This is because ImageNet pre-training determines the weights of backbones only, and the regularization effect of deep supervision remains even if the weights are fine-tuned. Although recent work [86, 18] emphasizes the effectiveness of normalization layers for training object detectors from scratch, it is worth exploring other forms of regularization including deep supervision [29, 24, 65].

He *et al.* [17, 18] found that "training longer for the first (large) learning rate is useful, but training for longer on small learning rates often leads to overfitting" on training Mask R-CNN. The increase of eigenspectrum in our results (Figure 4) with [70] can explain the overfitting as follows: (i) The learning rates for training object detectors decay. (ii) The detectors capture more detailed information about *training data* by finer optimization with the small learning rates. (iii) The eigenvalues and the intrinsic dimensionalities of the detectors increase. (iv) The bias decreases and the variance increases. (v) The detectors overfit if trained for longer on the small learning rates.

As described above, eigenspectrum dynamics are useful for analyzing which feature map is responsible for what information at which time. We believe that eigenspectrum dynamics can be a tool for analyzing neural architectures and learning rate schedules, or early stopping by predicting generalization error with eigenspectrum of training data.

5.3. Limitations and Weakness

We use ResNet and its variants, FPN, and Faster/Mask R-CNN in our experiments. It would be also interesting to conduct experiments with single-shot object detectors like SSD [41] and VGG-16 [67] without FPN. However, we believe that our analysis is meaningful for the computer vision community since Faster/Mask R-CNN are standard methods for object detection and instance segmentation.

Our method can only determine the widths of feature maps. Combining our method with compound scaling [73] and gradient-based NAS [38, 2, 80] to determine network depth, image resolution, and operations would give us further advantages.

We only consider MACs and the number of parameters as metrics of model efficiency. We should consider other metrics like memory footprint [60], memory access cost [47], and real latency on target platforms [81, 72, 77, 8].

Our method resets weights by random initialization. This choice is practical for complicated object detectors because it makes codes and experiments simple. However, applying pruning methods [30, 22, 45, 71] to object detectors may be better to train more efficient and accurate models.

We trained parameters after the determination of architectures in this paper. Considering the results of recent work [52], the simultaneous optimization of architectures and parameters is a highly important future direction, though the idea is classical (*e.g.*, TWEANNs; Topology and Weight Evolving Artificial Neural Networks). We believe that our analysis, method, and results are beneficial for the optimization since eigenspectrum is related to both architectures and parameters.

References

[1] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *ICML*, 2018.

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

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

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

[5] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. DetNAS: Backbone search for object detection. *arXiv:1903.10979*, 2019.

[6] Bowen Cheng, Yunchao Wei, Honghui Shi, Rogerio Feris, Jinjun Xiong, and Thomas Huang. Revisiting RCNN: On awakening the classification power of Faster RCNN. In *ECCV*, 2018.

[7] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016.

[8] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. ChamNet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019.

[9] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

[10] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *CVPR*, 2014.

[11] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019.

[12] Ross Girshick. Fast R-CNN. In *ICCV*, 2015.

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[14] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[15] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. MorphNet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

[16] 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:1706.02677*, 2017.

[17] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking ImageNet pre-training. *arXiv:1811.08883*, 2018.

[18] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking ImageNet pre-training. In *ICCV*, 2019.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

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

[21] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *ECCV*, 2018.

[22] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[23] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

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

[25] Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. What makes ImageNet good for transfer learning? *arXiv:1608.08614*, 2016.

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

[27] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 2017.

[28] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *ECCV*, 2018.

[29] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[31] Hengduo Li, Bharat Singh, Mahyar Najibi, Zuxuan Wu, and Larry S. Davis. An analysis of pre-training on object detection. *arXiv:1904.05871*, 2019.

[32] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-DSOD: Lightweight object detection for resource-restricted usages. In *BMVC*, 2018.

[33] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, 2017.

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

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

[36] 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.

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

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

[39] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *arXiv:1809.02165*, 2018.

[40] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018.

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

[42] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[43] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. MetaPruning: Meta learning for automatic neural network channel prunings. In *ICCV*, 2019.

[44] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, 2017.

[45] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[46] Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. Differentiable learning-to-normalize via switchable normalization. In *ICLR*, 2019.

[47] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

[48] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018.

[49] Francisco Massa and Ross Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark, 2018.

[50] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018.

[51] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. MegDet: A large mini-batch object detector. In *CVPR*, 2018.

[52] Siyuan Qiao, Zhe Lin, Jianming Zhang, and Alan Yuille. Neural Rejuvenation: Improving deep network training by enhancing computational resource utilization. In *CVPR*, 2019.

[53] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NIPS*, 2017.

[54] Maithra Raghu, Chiyuan Zhang, Jon M. Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. *arXiv:1902.07208*, 2019.

[55] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, real-time object detection. In *CVPR*, 2016.

[56] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *CVPR*, 2017.

[57] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[58] Michael T. Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G. Dietterich. To transfer or not to transfer. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, 2005.

[59] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[60] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[61] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *NeurIPS*, 2018.

[62] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *ICLR*, 2018.

[63] Shreyas Saxena and Jakob J. Verbeek. Convolutional neural fabrics. In *NIPS*, 2016.

[64] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[65] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. DSOD: Learning deeply supervised object detectors from scratch. In *ICCV*, 2017.

[66] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv:1703.00810*, 2017.

[67] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[68] Bharat Singh and Larry S. Davis. An analysis of scale invariance in object detection – SNIP. In *CVPR*, 2018.

[69] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017.

[70] Taiji Suzuki. Fast generalization error bound of deep learning from a kernel perspective. In *AISTATS*, 2018.

[71] Taiji Suzuki, Hiroshi Abe, Tomoya Murata, Shingo Horiuchi, Kotaro Ito, Tokuma Wachi, So Hirai, Masatoshi Yukishima, and Tomoaki Nishimura. Spectral-Pruning: Compressing deep neural network via spectral analysis. *arXiv:1808.08558*, 2018.

[72] 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.

[73] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[74] Shou-Yao Roy Tseng. Detectron.pytorch. `https://github.com/roytseng-tw/Detectron.pytorch`, 2018.

[75] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *CVPR*, 2019.

[76] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[77] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

[78] Yuxin Wu and Kaiming He. Group Normalization. In *ECCV*, 2018.

[79] Lingxi Xie and Alan Yuille. Genetic CNN. In *ICCV*, 2017.

[80] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *ICLR*, 2019.

[81] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018.

[82] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

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

[84] Amir R. Zamir, Alexander Sax, William Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *CVPR*, 2018.

[85] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *ICLR*, 2019.

[86] Rui Zhu, Shifeng Zhang, Xiaobo Wang, Longyin Wen, Hailin Shi, Liefeng Bo, and Tao Mei. ScratchDet: Exploring to train single-shot object detectors from scratch. In *CVPR*, 2019.

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

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