2021 IEEE International Conference on Multimedia and Expo (ICME) | 978-1-6654-3864-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICME51207.2021.9428105

COCONV: LEARNING DYNAMIC COOPERATIVE CONVOLUTION FOR IMAGE RECOGNITION

Kien X. Nguyen[1], *Tiffany Ryu*[2], *Jocelyn Zhang*[2], *Xu Ma*[2], *Qing Yang*[2], *Song Fu*[2], *Paparao Palacharla*[3], *Nannan Wang*[3], *Xi Wang*[3]

[1]Texas Christian University, [2]University of North Texas, [3]Fujitsu Network Communications
k.x.nguyen@tcu.edu; Tiffanyryu2020@gmail.com; {jocelynzhang, xuma}@my.unt.edu;
{qing.yang, Song.Fu}@unt.edu; { Paparao.Palacharla, Nannan.Wang, Xi.Wang}@fujitsu.com

ABSTRACT

In this paper, we present a conceptually simple, yet powerful method for image recognition. The method, called Cooperative Dynamic Convolution (CoConv), introduces a cooperative learning of dynamic convolution from multiple convolutional experts. CoConv can be used as a substitute for the traditional static convolution, and can be seamlessly integrated in various visual models. Moreover, CoConv is easy to train with only a minimal computational overhead introduced in the inference phase. CoConv is trained by using multiple convolutional experts simultaneously, and the convolutional weights are merged by a weighted summation before convolutional operations for efficiency during inference. Results from extensive experiments show that CoConv leads to consistent improvement for image classification on various datasets, independent of the choice of the base convolutional network. Remarkably, CoConv improves the top-1 classification accuracy of ResNet18 by **3.06%** on ImageNet. The code is available at: https://github.com/Nyquixt/CoConv.

Index Terms— Convolutional Neural Networks, Dynamic Convolution, Cooperative Convolutional Experts, Image Recognition, Computer Vision.

1. INTRODUCTION

The past few years has witnessed a dramatic performance improvement of convolutional neural networks (CNNs) in computer vision, multimedia, and virtual reality. To further enhance the capability of CNNs, more and more layers and/or design mechanisms are used, which make CNNs more complicated, such as VGG16 [1], ResNet [2] and DenseNet [3]. With more learnable layers added, a model is able to extract deeper features, exhibiting powerful representation ability. Meanwhile, mechanisms, such as attention mechanism [4, 5], are also explored in CNN design. By adding a lightweight attention module, a CNN model is able to emphasize important features while suppressing trivial ones, which further improves the performance. While both strategies are instrumental, there is a lack of deep delving into the construction of learnable weights in convolutional layers, which we observe to be imperative for the performance of CNNs.

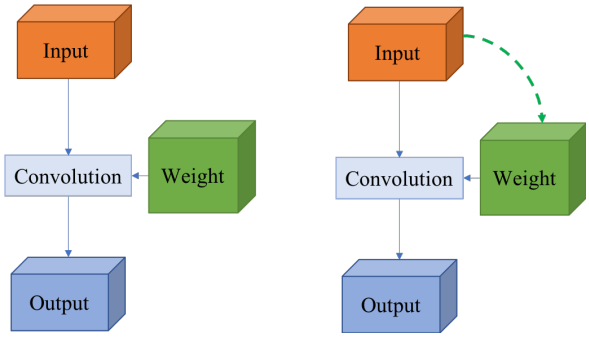

Fig. 1: **Difference between static (*Left*) and dynamic (*Right*) convolutions.** In static (traditional) convolution, the weights and input images are mutually uncorrelated. In dynamic convolution, the weights are effected or partially affected by the input images.

In a traditional CNN model, the weights in each learnable layer are learnt from the entire training set, and fixed after the training phase. *Namely, in the course of inference, the static weights in pre-trained models and the testing images are uncorrelated*. While the property of uncorrelatedness could be considered as generalization, we emphasize that a dynamic weight construction, where the weights are conditionally related to the input images, is more adaptive and tailored to achieve better performance with a minimal extra cost. Figure. 1 illustrates the difference.

In this paper, we present a Dynamic Cooperative Convolution, (CoConv), where a traditional convolution is disentangled into several standard convolutional experts and the weights are partially dependent on the input feature maps. To adaptively and dynamically combine the weights in convolutional experts, we further design a lightweight channel-wise routing module, which generates a proper weight for each channel in every expert based on the input features. Note that the training and inference phrases of our CoConv are

slightly different. In the training phase, all the experts are trained jointly, and cooperatively generate the feature maps for the next layer. CoConv enjoys the advantages of over-parameterization [6]. Notice that such a manner would introduce more parameters and computations in the training phrase, but it is not the main concern in our design (and for most applications). We focus on the inference efficiency and the performance improvements, which are the key points for most scenarios. In the inference phrase, CoConv dynamically computes the channel-wise weights for each expert, and combines the weights in all convolutional experts before convolutional operations to reduce redundant computations. As a result, a minimal amount of add-on operations, i.e. only the FLOPs and parameters in calculating the channel-wise weights, are introduced to the networks. We emphasize that the negligible additional computation overhead required by our CoConv is justified by its significant improvement of the model performance.

By introducing the novel CoConv, a CNN model is able to significantly improve its capability of feature representation while maintaining a similar inference complexity. Results from extensive experiments on CIFAR-100, Tiny-ImageNet and ImageNet benchmarks have demonstrated the effectiveness of CoConv. Empirical studies on CIFAR-100 and Tiny-ImageNet datasets have shown that CoConv outperforms existing models when being used on various base models. Remarkably, CoConv improves the top-1 classification accuracy of ResNet18 by 3.06% (72.70% *vs.* 69.64%) on the large-scale ImageNet dataset.

2. RELATED WORK

Deep Convolutional Neural Networks. In the past decade, deep convolutional neural networks have shown promising results in many fields, like computer vision, multimedia technology, medical testing, *etc*. One of the earlier models, AlexNet [7], employs 5 convolutional layers and 3 fully connected layers to extract visual concepts. By using 3×3 filters, VGG [1] employs 16 learnable layers for feature extraction, providing a deeper feature representation. However, as CNNs go deeper, a critical issue arises, that is with an increasing network depth, the performance gets saturated and sometimes even degraded in training and inference [2]. To mitigate this issue, ResNet [2] employs shortcut connections and DenseNet [3] connects each layer with every other layer, to facilitate information to flow between layers.

Although these deeper networks may achieve a better accuracy, they tend to be very complex and computationally expensive. Consequently, researchers are developing smaller, more efficient networks. MobileNet [8, 9] achieves this goal through the use of depth-wise separable convolutions. Another efficient network, ShuffleNet [10], uses point-wise group convolutions and channel shuffle to reduce computation cost while preserving a comparable accuracy. Recently, neural architecture searching has shown promising results in network design [11, 12].

Multi-Branch Operations. Since Inception [13], a myriad of multi-branch networks have been developed, each aggregating features generated by multiple branches to produce a compositive representation. An example of a network with a multi-branch architecture is ResNet [2] which consists of two branches. The output of the shortcut connection branch that performs identity mapping is combined with the output of the other branch. Built on ResNet's structure, ResNeXt [14] increases the cardinality (the set of transformations) and uses grouped convolutions whose outputs are concatenated to produce a final output. Differently, Inception series [13, 15] consider the feature aggregation by multiple heterogeneous branches. Each Inception module consists of multiple branches with different filter sizes as well as a max pooling branch whose outputs are concatenated.

In this paper, we leverage the advantages of multiple branches for enhanced feature representations, while introducing a minimal computation overhead.

Dynamic Convolution Unlike static convolutional operations where the kernels are fixed, the kernels in dynamic convolutions are closely related to the inputs, leading to flexible yet favorable convolutional operations. An example of dynamic convolution is CondConv [16], which leverages multiple convolutional paths to extract discriminative features, and employs a lightweight attention module to dynamically adjust the weights of each path based on the input features. DY-CNN [17] explores a same strategy and replaces the Sigmoid function with Softmax to constrain that the sum of attention outputs should be 1. Different from the existing works, our method follows the stream and provides a more delicate design by leveraging the channel-wise operation and a multi-scale feature representation.

3. DYNAMIC CONVOLUTION

The framework of conditionally parameterized convolution (CondConv) [16] consists of *n* convolution experts, scaled by an n-dimensional routing vector $\alpha \in \mathbb{R}^n$, which is dependent on the input features and helps make the convolution layer dynamic. Let the weights in n convolution experts be $\mathbf{W}_1$, $\mathbf{W}_2$, ..., $\mathbf{W}_n \in \mathbb{R}^{C' \times C \times k_h \times k_w}$, where C' is the number of out channels, C is the number of the feature channels and $k_h \times k_w$ indicates the kernel shape. The convolutional weights in each expert are scaled by the corresponding attention value, and then aggregated as the final unified weights, which can be formulated as

$$\mathbf{W} = \sum_{i=1}^{n} \alpha_i \mathbf{W}_i. \tag{1}$$

Equation (1) builds the foundation for convolution fusion. Let $\circledast$ denote the convolutional operation; the traditional con-

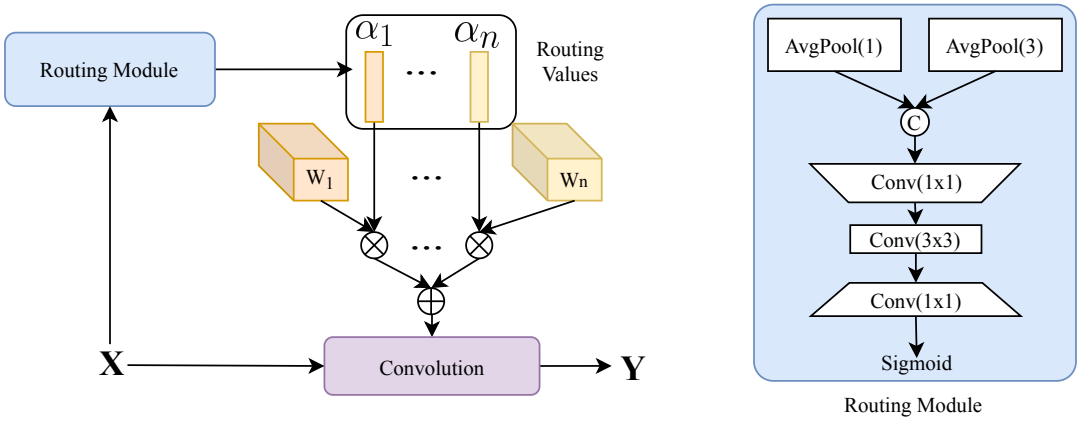

Fig. 2: The framework of CoConv. The routing module is depicted on the right. $\otimes$ represents element-wise multiplication, $\oplus$ denotes summation and © is concatenation.

volution can be presented as $\mathbf{Y} = \mathbf{W} \circledast \mathbf{X}$, where $\mathbf{Y}$ is the output and $\mathbf{X}$ is the input. Considering the linearity of a convolutional operation, we can rewrite the convolutional operation as follows.

$$\begin{aligned} \mathbf{Y} &= \mathbf{W} \circledast \mathbf{X} \\ &= \left(\sum_{i=1}^{n} \alpha_i \mathbf{W}_i \right) \circledast \mathbf{X} \\ &= \sum_{i=1}^{n} \left(\alpha_i \mathbf{W}_i \circledast \mathbf{X} \right). \end{aligned} \tag{2}$$

That is, the traditional convolutional operation theoretically can be split to multiple parallel convolutional operations. Such an property makes it possible to train multiple convolutional layers in parallel, and fuse them during inference for better efficiency [17, 18, 16].

Note that the only difference between CondConv and DY-CNN is the generation of the routing values α for convolutional experts. However, we emphasize that the capacity of dynamic convolution is not fully explored by such a coarse (i.e. one real-value for each expert and high-level implementation) routing design. Inspired by this, we design CoConv, which incorporates a more elaborate design of dynamic convolution, to achieve better flexibility and higher performance.

4. COOPERATIVE DYNAMIC CONVOLUTION

Given an intermediate feature map $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$, where N is the batch size, C denotes the number of channels and $H \times W$ represents the spatial size, CoConv calculates channel-wise recalibration weights for n convolutional experts based on the multi-scaled representation of $\mathbf{X}$. With the recalibration weights, all experts dynamically generate a synthetic output in a cooperative fashion. Figure. 2 presents the design of our CoConv. Next, we detail each component of CoConv.

4.1. Multi-scale Feature Extraction

We re-scale a feature map $\mathbf{X}$ by average pooling to extract multi-scale feature representations.

Existing studies on multiple scales [19, 20, 21] suffer from a discrepancy between efficiency and performance. To address this problem, we use two average poolings to extract the global and coarse feature representations.

Specifically, we consider two scales in our design, that is a 1×1 resolution to extract the global context and a 3×3 resolution to present a patch-wise coarse representation. The two average poolings are deployed in parallel and executed at the same time. We next expand the global context to a size of 3×3 to match the resolution. The two outputs are concatenated to produce a new feature map $\mathbf{X}' \in \mathbb{R}^{N \times 2C \times 3 \times 3}$, where $2C$ is the number of channels in the generated feature map. By doing so, we efficiently extract multi-scale feature representations with a negligible computation overhead.

4.2. Depth-wise Separable Block

After the multi-scale feature representation $\mathbf{X}'$ is extracted, we calculate the recalibration weights by using a depth-wise separable block. We aim to achieve an embedding $f(\mathbf{X}')$, from a feature map $\mathbf{X}'$ to a recalibration weight α, such that each channel in a convolutional expert can be dynamically recalibrated based on the input. To achieve this goal, we first squeeze the channel dimension of $\mathbf{X}'$ by a factor of 16 using a point-wise convolution, which significantly reduces the computation complexity of embedding $f(\mathbf{X}')$. To fully leverage the advantages of multi-scale representations, we next gather the spatial information by using a learnable linear combination for each channel. After that, the embedding vector is transformed to a new vector α with a channel dimension

$n \times C'$, where n is the number of convolutional experts and C' is the output channel dimension. We reshape α and spilt it to n sub-vectors, where each one provides channel-wise routing weight for the corresponding convolutional expert.

4.3. Channel-wise Attention

In order to enhance a model's generalization and performance, we employ the attention mechanism on the channel dimension of each expert, instead of scaling the entire expert by the same weight. For each $\alpha_i \in \mathbb{R}^{C'}$, we recalibrate the weight $\mathbf{W}_i$ of the ith convolutional expert by computing $\sigma(\alpha_i) \otimes \mathbf{W}_i$, where $\otimes$ denotes element-wise multiplication. We choose the sigmoid function $\sigma(\cdot)$ to scale the routing values to a range of $(0, 1)$. Hence, the fused convolution operation during inference is expressed as

$$\mathbf{Y} = \left(\sum_{i=1}^{n} \sigma(\alpha_i) \otimes \mathbf{W}_i \right) \circledast \mathbf{X}. \quad (3)$$

In our implementation, we set the number of experts n to 4 by default. In addition, we include a batch normalization (BN) layer in each convolutional expert to achieve a better feature representation.

5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our CoConv for image recognition tasks on three datasets: CIFAR-100 [22], Tiny-ImageNet [23], and ImageNet [24]. To study the generalization of CoConv, we integrate it into multiple base CNN architectures, including AlexNet [7], ResNet [2], and MobileNetV2 [9]. All our implementations are based on PyTorch [25]. We have conducted extensive experiments and ablation studies.

5.1. Performance on CIFAR-100 and Tiny-ImageNet

In this set of experiments, we compare the performance of CoConv with that of DY-CNN [17] and CondConv [16] (which also include dynamic convolution) on CIFAR-100 and Tiny-ImageNet.

CIFAR-100 consists of 60K images (each with 32×32 pixels) belonging to 100 classes with 600 images in each class. Tiny-ImageNet includes all the images from ImageNet in a lower resolution (64 × 64 pixels). We train each model on the two datasets using the stochastic gradient descent method with a momentum as 0.9 and and a weight decaying factor as $5e^{-4}$. Each model is trained for 90 epochs with a mini-batch size of 128. We initialize the learning rate as 0.01, and decrease it by a factor of 10 every 30 epochs. Following the practice in [2], we horizontally flip the images with a probability of 50%. Table 1 and Table 2 present the results.

In the tables, we can see our CoConv consistently outperforms other methods on different base CNN architectures,

Table 1: Classification performance (%) on CIFAR-100. The number of experts n is 4.

	AlexNet	ResNet18	MobileNetV2
Original	58.63	71.87	69.10
CondConv	58.49	72.90	70.37
DY-CNN	58.42	73.08	69.32
CoConv	**58.98**	**73.48**	**70.89**

Table 2: Classification performance (%) on Tiny-ImageNet. The number of experts n is 4.

	AlexNet	ResNet18	MobileNetV2
Original	51.13	61.33	59.30
CondConv	51.51	61.19	**60.58**
DY-CNN	51.19	61.61	59.87
CoConv	**51.93**	**63.25**	60.42

achieving the best result in 5 out of 6 settings. Specifically, CoConv outperforms the original ResNet18, CondConv, and DY-CNN counterparts by 1.92, 2.06 and 1.64, respectively. The results infer that multi-scale feature extraction and channel-wise attention introduced to the routing function contribute to accuracy improvement, particularly for bigger datasets with a larger input spatial size and more complex features to generalize.

5.2. Experiments on ImageNet

In another set of experiments, we evaluate CoConv's performance on ImageNet [24] which is a widely used, large-scale image classification benchmark, containing 1.28M training images and 50K images for validation in 1000 categories. In addition to DY-CNN and CondConv, we include several plug-in modules in our comparison, such as SENet [4], GENet [5], and ACNet [18].

We train each model on the ImageNet training set and measure its single-crop (224×224 pixels) top-1 and top-5 accuracy on the validation set. Each model is trained for 100 epochs with 32 images per GPU (256 in a batch). The learning rate is set to 0.1 initially, and decreased by a factor of 10 every 30 epochs. For a fair comparison, the number of experts n is 3 for DY-CNN, CondConv, and our CoConv, and BN layer is not included in each expert. We adopt the common-used ResNet18 as the base model.

Table 3 lists the experimental results. We can see CoConv achieves a significant improvement over the base ResNet18 (about **3.06%**), and outperforms all the competitors by a clear margin. More specifically, when compared with other dynamic convolution methods, our CoConv surpasses CondConv and DY-CNN by 0.76% and 2.08% respectively. We note CoConv introduces some more FLOPs and parameters

Table 3: Single-cropped classification performance (%) on the ImageNet validation set.

Models	Multi-conv	Type	Top-1 acc.	Top-5 acc.	FLOPs(G)	Parameters(M)
ResNet18 [2]		static	69.635	89.005	1.822	11.690
SE-ResNet18 [4]		static	71.024	89.916	1.823	11.779
GE-ResNet18 [5]		static	70.405	89.778	1.825	11.753
AC-ResNet18 [18]	✓	static	70.779	89.676	**1.822**	**11.690**
CondConv-ResNet18 [16]	✓	dynamic	71.935	90.239	1.822	11.701
DY-CNN-ResNet18 [17]	✓	dynamic	70.613	89.684	1.822	12.001
CoConv-ResNet18 (ours)	✓	dynamic	**72.696**	**90.717**	1.824	12.166

Table 4: Impact of the number of experts n on the performance of CoConv on Tiny ImageNet.

Base model	$n=2$	$n=4$	$n=8$
CoConv-AlexNet	**52.14**	51.93	52.00
CoConv-ResNet18	63.08	**63.25**	62.57
CoConv-MobileNetV2	60.11	**60.42**	59.75

during inference. However, the small additional computational overhead, i.e. 0.002G FLOPs and 0.476M parameters, from CoConv is justified by its remarkable classification accuracy, i.e. 72.70% *vs.* 69.64%.

5.3. Ablation Studies

We conduct a series of ablation studies to analyze the internal structure and operations of CoConv.

The Number of Experts. We first investigate how the number of experts n affects the performance of CoConv. We conduct the experiments on the Tiny ImageNet with a configuration similar to that in Section 5.1. Table 4 presents the results which show the best performance is achieved when the default value $n = 4$ is applied. We note when a simpler architecture, such as AlexNet, is used, the accuracy may be higher when $n = 2$. Moreover, the number of experts affects the computational overhead in training (i.e. more experts lead to more FLOPs and parameters), but not in inference.

Input Channels vs. Output Channels. We also study the effect of applying the vector α on the input and output channel dimensions of the convolutional experts. We test the three backbone architectures AlexNet, ResNet18 and MobileNetV2 on on CIFAR-100 and Tiny ImageNet. Table 5 lists the results. We can see applying attention on the output channel dimension (as implemented in our default architecture) helps CoConv perform better, especially on deeper CNN architectures that include more convolutional layers.

Single Scale vs. Multi Scale. We demonstrate the effectiveness of the multi-scaled feature extraction in our model. To make the routing module single-scaled, we remove the average pooling layer of 3×3 and transform the 3×3 Conv layer to 1×1. We can see from Table 6 that the multi-scaled version of CoConv yields better results overall, specifically on more complex backbones and on larger datasets.

Table 5: Performance (%) comparison between applying the attention vector to the input (In) and output (Out) channel dimensions of convolutional experts in CoConv.

		CIFAR-100		Tiny ImageNet	
		$n=2$	$n=4$	$n=2$	$n=4$
AlexNet	In	**59.93**	**60.00**	**52.36**	51.78
	Out	59.46	58.98	52.14	**51.93**
ResNet18	In	72.52	73.29	61.67	62.15
	Out	**73.12**	**73.48**	**63.08**	**63.25**
MobileNetV2	In	69.85	69.61	**60.22**	60.00
	Out	**70.76**	**70.89**	60.11	**60.42**

Table 6: Performance (%) comparison between single-scaled (Single) and mutil-scaled (Multi) feature extraction in CoConv's attention module, with $n = 4$.

		CIFAR-100	Tiny ImageNet
AlexNet	Single	**59.38**	51.57
	Multi	58.98	**51.93**
ResNet18	Single	72.27	62.17
	Multi	**73.48**	**63.25**
MobileNetV2	Single	70.06	60.13
	Multi	**70.89**	**60.42**

6. CONCLUSIONS

In this paper, we describe a novel method, CoConv, which enhances the feature representational ability of convolutions by dynamically and cooperatively learning the representations from multiple convolutional experts in parallel. CoConv incurs a little more computation during training, but only a minimal amount of extra computation in inference. Experimen-

tal results on CIFAR-100, Tiny-ImageNet, and ImageNet all show the outstanding performance of CoConv. In particular, CoConv improves the the top-1 accuracy of ResNet18 by 3.06% on ImageNet. This significant performance improvement makes CoConv promising for various multimedia applications in practice.

7. ACKNOWLEDGMENT

This work has been supported in part by the National Science Foundation grants CNS-1852134, OAC-2017564, ECCS-2010332, CNS-2037982, and CNS-1563750. We thank the anonymous reviewers for their constructive comments, which helped us improve this paper.

8. REFERENCES

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

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

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 4700–4708.

[4] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu, "Squeeze-and-excitation networks," 2019.

[5] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi, "Gather-excite: Exploiting feature context in convolutional neural networks," in *NeurIPS*, 2018.

[6] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song, "A convergence theory for deep learning via over-parameterization," in *ICML*. PMLR, 2019, pp. 242–252.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012, pp. 1097–1105.

[8] 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 preprint arXiv:1704.04861*, 2017.

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

[10] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, 2018.

[11] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang, "Model rubik's cube: Twisting resolution, depth and width for tinynets," *NeurIPS*, vol. 33, 2020.

[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.

[13] 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, pp. 1–9.

[14] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016, pp. 2818–2826.

[16] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam, "Condconv: Conditionally parameterized convolutions for efficient inference," in *NeurIPS*, 2019.

[17] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu, "Dynamic convolution: Attention over convolution kernels," in *CVPR*, 2020, pp. 11030–11039.

[18] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," in *ICCV*, 2019, pp. 1911–1920.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[20] Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip HS Torr, "Res2net: A new multi-scale backbone architecture," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[21] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al., "Deep high-resolution representation learning for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[22] Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[23] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," 2015.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.