CondConv: Conditionally Parameterized Convolutions for Efficient Inference

Brandon Yang*
Google Brain
bcyang@google.com

Gabriel Bender
Google Brain
gbender@google.com

Quoc V. Le
Google Brain
qvl@google.com

Jiquan Ngiam
Google Brain
jngiam@google.com

Abstract

Convolutional layers are one of the basic building blocks of modern deep neural networks. One fundamental assumption is that convolutional kernels should be shared for all examples in a dataset. We propose conditionally parameterized convolutions (CondConv), which learn specialized convolutional kernels for each example. Replacing normal convolutions with CondConv enables us to increase the size and capacity of a network, while maintaining efficient inference. We demonstrate that scaling networks with CondConv improves the performance and inference cost trade-off of several existing convolutional neural network architectures on both classification and detection tasks. On ImageNet classification, our CondConv approach applied to EfficientNet-B0 achieves state-of-the-art performance of 78.3% accuracy with only 413M multiply-adds. Code and checkpoints for the CondConv Tensorflow layer and CondConv-EfficientNet models are available at: `https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/condconv`.

1 Introduction

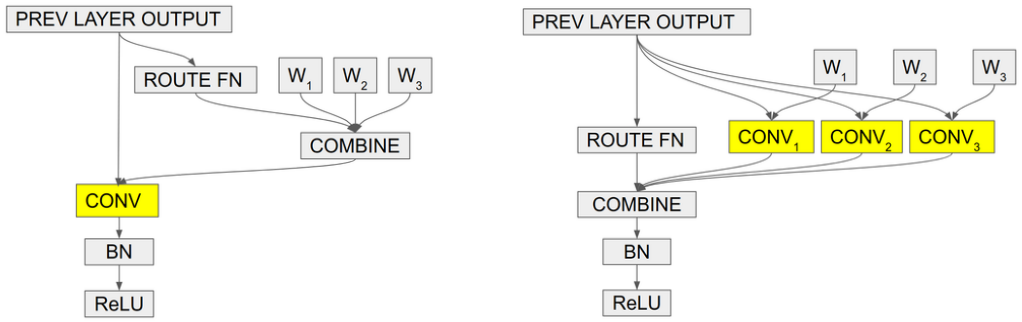
Deep convolutional neural networks (CNNs) have achieved state-of-the-art performance on many tasks in computer vision [23, 22]. Improvements in performance have largely come from increasing model size and capacity to scale to larger and larger datasets [29, 17, 33]. However, current approaches to increasing model capacity are computationally expensive. Deploying the best-performing models for inference can consume significant datacenter capacity [19] and are often not feasible for applications with strict latency constraints.

One fundamental assumption in the design of convolutional layers is that the same convolutional kernels are applied to every example in a dataset. To increase the capacity of a model, model developers usually add more convolutional layers or increase the size of existing convolutions (kernel height/width, number of input/output channels). In either case, the computational cost of additional capacity increases proportionally to the size of the input to the convolution, which can be large.

Due to this assumption and focus on mobile deployment, current computationally efficient models have very few parameters [15, 36, 41]. However, there is a growing class of computer vision applications that are not constrained by parameter count, but have strict latency requirements at inference, such as real-time server-side video processing and perception for self-driving cars. In this paper, we aim to design models to better serve these applications.

*Work done as part of the Google AI Residency.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.



(a) CondConv: $(\alpha_1 W_1 + \ldots + \alpha_n W_n) * x$ (b) Mixture of Experts: $\alpha_1(W_1 * x) + \ldots + \alpha_n(W_n * x)$

Figure 1: (a) Our CondConv layer architecture with $n = 3$ kernels vs. (b) a mixture of experts approach. By parameterizing the convolutional kernel conditionally on the input, CondConv is mathematically equivalent to the mixture of experts approach, but requires only 1 convolution.

We propose conditionally parameterized convolutions (CondConv), which challenge the paradigm of static convolutional kernels by computing convolutional kernels as a function of the input. In particular, we parameterize the convolutional kernels in a CondConv layer as a linear combination of n experts $(\alpha_1 W_1 + \ldots + \alpha_n W_n) * x$, where $\alpha_1, \ldots, \alpha_n$ are functions of the input learned through gradient descent. To efficiently increase the capacity of a CondConv layer, model developers can increase the number of experts. This is much more computationally efficient than increasing the size of the convolutional kernel itself, because the convolutional kernel is applied at many different positions within the input, while the experts are combined only once per input. This allows model developers to increase model capacity and performance while maintaining efficient inference.

CondConv can be used as a drop-in replacement for existing convolutional layers in CNN architectures. We demonstrate that replacing convolutional layers with CondConv improves model capacity and performance on several CNN architectures on ImageNet classification and COCO object detection, while maintaining efficient inference. In our analysis, we find that CondConv layers learn semantically meaningful relationships across examples to compute the conditional convolutional kernels.

2 Related Work

Conditional computation. Similar to CondConv, conditional computation aims to increase model capacity without a proportional increase in computation cost. In conditional computation models, this is achieved by activating only a portion of the entire network for each example [3, 8, 5, 2]. However, conditional computation models are often challenging to train, since they require learning discrete routing decisions from individual examples to different experts. Unlike these approaches, CondConv does not require discrete routing of examples, so can be easily optimized with gradient descent.

One approach to conditional computation uses reinforcement learning or evolutionary methods to learn discrete routing functions [34, 30, 25, 9]. BlockDrop [47] and SkipNet [45] use reinforcement learning to learn the subset of blocks needed to process a given input. Another approach uses unsupervised clustering methods to partition examples into sub-networks. Gross et al. [12] use a two stage training and clustering pipeline to train a hard mixture of experts model. Mullapudi et al. [31] use clusters as labels to train a routing function between branches in a deep CNN model. Finally, Shazeer et al. [37] proposed the sparsely-gated mixture-of-experts layer, which which achieves significant success on large language modeling using noisy top-k gating.

Prior work in computation demonstrates the potential of designing large models that process different sets of examples with different sub-networks. Our work on CondConv pushes the boundaries of this paradigm, by enabling each individual example to be processed with different weights.

Weight generating networks. Ha et al. [13] propose the use of a small network to generate weights for a larger network. Unlike CondConv, for CNNs, these weights are the same for every example in the dataset. This enables greater weight-sharing, which achieves lower parameter count but worse performance than the original network. In neural machine translation, Platanios et al. [32] generate

weights to translate between different language pairs, but use the same weights for every example within each language pair.

Multi-branch convolutional networks. Multi-branch architectures like Inception [40] and ResNext [48] have shown success on a variety of computer vision tasks. In these architectures, a layer consists of multiple convolutional branches, which are aggregated to compute the final output. A CondConv layer is mathematically equivalent to a multi-branch convolutional layer where each branch is a single convolution and outputs are aggregated by a weighted sum, but only requires the computation of one convolution.

Example dependent activation scaling. Some recent work proposes to adapt the activations of neural networks conditionally on the input. Squeeze-and-Excitation networks [16] learn to scale the activations of every layer output. GaterNet [4] uses a separate network to select a binary mask for filters for a larger backbone network. Attention-based methods [28, 1, 44] scale previous layer inputs based on learned attention weights. Scaling activations has similar motivations as CondConv, but is restricted to modulating activations in the base network.

Input-dependent convolutional layers. In language modeling, Wu et al. [46] use input-dependent convolutional kernels as a form of local attention. In vision, Brabandere et al. [18] generate small input-dependent convolutional filters to transform images for next frame and stereo prediction. Rather than learning input-dependent weights, Dai et al. [7] propose to learn different convolutional offsets for each example. Finally, in recent work, SplineNets [20] apply input-dependent convolutional weights, modeled as 1-dimensional B-splines, to implement continuous neural decision graphs.

3 Conditionally Parameterized Convolutions

In a regular convolutional layer, the same convolutional kernel is used for all input examples. In a CondConv layer, the convolutional kernel is computed as a function of the input example (Fig 1a). Specifically, we parameterize the convolutional kernels in CondConv by:

$$Output(x) = \sigma((\alpha_1 \cdot W_1 + \ldots + \alpha_n \cdot W_n) * x)$$

where each $\alpha_i = r_i(x)$ is an example-dependent scalar weight computed using a routing function with learned parameters, n is the number of experts, and σ is an activation function. When we adapt a convolutional layer to use CondConv, each kernel W_i has the same dimensions as the kernel in the original convolution.

We typically increase the capacity of a regular convolutional layer by increasing the kernel height/width or number of input/output channels. However, each additional parameter in a convolution requires additional multiply-adds proportional to the number of pixels in the input feature map, which can be large. In a CondConv layer, we compute a convolutional kernel for each example as a linear combination of n experts before applying the convolution. Crucially, each convolutional kernel only needs to be computed once but is applied at many different positions in the input image. This means that by increasing n, we can increase the capacity of the network with only a small increase in inference cost; each additional parameter requires only 1 additional multiply-add.

A CondConv layer is mathematically equivalent to a more expensive linear mixture of experts formulation, where each expert corresponds to a static convolution (Fig 1b):

$$\sigma((\alpha_1 \cdot W_1 + \ldots + \alpha_n \cdot W_n) * x) = \sigma(\alpha_1 \cdot (W_1 * x) + \ldots + \alpha_n \cdot (W_n * x))$$

Thus, CondConv has the same capacity as a linear mixture of experts formulation with n experts, but is computationally efficient since it requires computing only one expensive convolution. This formulation gives insight into the properties of CondConv and relates it to prior work on conditional computation and mixture of experts. The per-example routing function is crucial to CondConv performance: if the learned routing function is constant for all examples, a CondConv layer has the same capacity as a static convolutional layer.

We wish to design a per-example routing function that is computationally efficient, able to meaningfully differentiate between input examples, and is easily interpretable. We compute the example-dependent routing weights $\alpha_i = r_i(x)$ from the layer input in three steps: global average pooling, fully-connected layer, Sigmoid activation.

$$r(x) = \text{Sigmoid}(\text{GlobalAveragePool}(x)\ R)$$

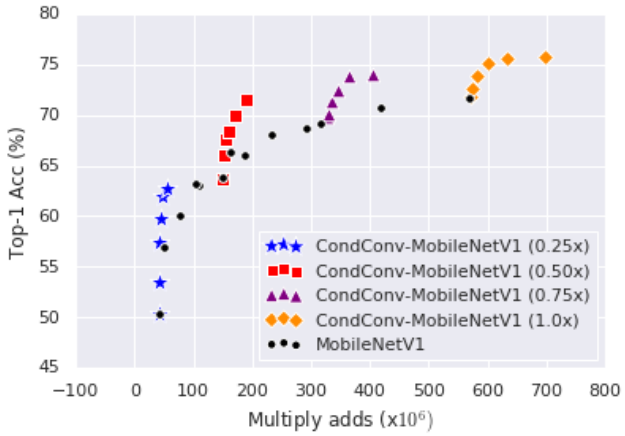


Figure 2: On ImageNet validation, increasing the number of experts per layer of our CondConv-MobileNetV1 models improves performance relative to inference cost compared to the MobileNetV1 frontier [38] across a spectrum of model sizes. Models with more experts per layer achieve monotonically higher accuracy. We train CondConv models with {1, 2, 4, 8, 16, 32} experts at width multipliers {0.25, 0.50, 0.75, 1.0}.

where R is a matrix of learned routing weights mapping the pooled inputs to n expert weights. A normal convolution operation operates only over local receptive fields, so our routing function allows adaptation of local operations using global context.

The CondConv layer can be used in place of any convolutional layer in a network. The same approach can easily be extended to other linear functions like those in depth-wise convolutions and fully-connected layers.

4 Experiments

We evaluate CondConv on ImageNet classification and COCO object detection by scaling up the MobileNetV1 [15], MobileNetV2 [36], ResNet-50 [14], MnasNet [41], and EfficientNet [42] architectures. In practice, we have two options to train CondConv models, which are mathematically equivalent. We can either first compute the kernel for each example and apply convolutions with a batch size of one (Fig. 1a), or we can use the linear mixture of experts formulation (Fig. 1b) to perform batch convolutions on each branch and sum the outputs. Current accelerators are optimized to train on large batch convolutions, and it is difficult to fully utilize them for small batch sizes. Thus, with small numbers of experts (<=4), we found it to be more efficient to train CondConv layers with the linear mixture of experts formulation and large batch convolutions, then use our efficient CondConv approach for inference. With larger numbers of experts (>4), training CondConv layers directly with batch size one is more efficient.

4.1 ImageNet Classification

We evaluate our approach on the ImageNet 2012 classification dataset [35]. The ImageNet dataset consists of 1.28 million training images and 50K validation images from 1000 classes. We train all models on the entire training set and compare the single-crop top-1 validation set accuracy with input image resolution 224x224. For MobileNetV1, MobileNetV2, and ResNet-50, we use the same training hyperparameters for all models on ImageNet, following [21], except we use BatchNorm momentum of 0.9 and disable exponential moving average on weights. For MnasNet [41] and EfficientNet [42], we use the same training hyperparameters as the original papers, with the batch size, learning rate, and training steps scaled appropriately for our hardware configuration. For fair comparison, we retrain all of our baseline models with the same hyperparameters and regularization search space as the CondConv models. We used accuracy on the validation set to determine early stopping. We measure performance as ImageNet top-1 accuracy relative to computational cost in multiply-adds (MADDs).

Table 1: ImageNet validation accuracy and inference cost for our CondConv models on several baseline model architectures. All models use 8 experts per CondConv layer. CondConv improves the accuracy of all baseline architectures with small relative increase in inference cost (<10%).

	Baseline		CondConv	
	MADDs ($\times 10^6$)	Top-1 (%)	MADDs ($\times 10^6$)	Top-1 (%)
MobileNetV1 (1.0x)	567	71.9	600	73.7
MobileNetV2 (1.0x)	301	71.6	329	74.6
MnasNet-A1	312	74.9	325	76.2
ResNet-50	4093	77.7	4213	78.6
EfficientNet-B0	391	77.2	413	78.3

For each baseline architecture, we evaluate CondConv by replacing convolutional layers with CondConv layers, and increasing the number of experts per layer. We share routing weights between layers in a block (a residual block, inverted bottleneck block, or separable convolution). Additionally, for some models, we replace the fully-connected classification layer with a 1x1 CondConv layer. For the exact architectural details, refer to Appendix A. Our ablation experiments in Table 3 and Table 4 suggest CondConv improves performance across a wide range of layer and routing architectures.

We use two general regularization techniques for models with large capacity. First, we use Dropout [39] on the input to the fully-connected layer preceding the logits, with keep probability between 0.6 and 1.0. Second, we also add data augmentation using the AutoAugment [6] ImageNet policy and Mixup [49] with $\alpha = 0.2$. To address overfitting in the large ResNet models, we additionally introduce a new data augmentation technique for CondConv based on Shake-Shake [10] by randomly dropping out experts during training.

On MobileNetV1, we find that increasing the number of CondConv experts improves accuracy relative to inference cost compared to the performance frontier with static convolutional scaling techniques using the channel width and input size (Figure 2). Moreover, we find that increasing the number of CondConv experts leads to monotonically increasing performance with sufficient regularization.

We further find that CondConv improves performance relative to inference cost on a wide range of architectures (Table 1). This includes architectures that take advantage of architecture search [42, 41], Squeeze-and-Excitation [16], and large architectures with ordinary convolutions not optimized for inference time [14]. For more in depth comparisons, see Appendix A.

Our CondConv-EfficientNet-B0 model achieves state-of-the-art performance of 78.3% accuracy with 413M multiply-adds, when compared to the MixNet frontier [43]. To directly compare our CondConv scaling approach to the compound scaling coefficient proposed by Tan et al. [42], we additionally scale the CondConv-EfficientNet-B0 model with a depth multiplier of 1.1x, which we call CondConv-EfficientNet-B0-depth. Our CondConv-EfficientNet-B0-depth model achieves 79.5% accuracy with only 614M multiply-adds. When trained with the same hyperparameters and regularization search space, the EfficientNet-B1 model, which is scaled from the EfficientNet-B0 model using the compound coefficient, achieves 79.2% accuracy with 700M multiply-adds. In this regime, CondConv scaling outperforms static convolutional scaling with the compound coefficient.

4.2 COCO Object Detection

We next evaluate the effectiveness of CondConv on a different task and dataset with the COCO object detection dataset [24]. Our experiments use the MobileNetV1 feature extractor and the Single Shot Detector [26] with 300x300 input resolution (SSD300).

Following Howard et al. [15], we train on the combined COCO training and validation sets excluding 8,000 minival images, which we evaluate our networks on. We train our models using a batch size

[2]Our re-implementation of the baseline models and our CondConv models use the same hyperparameters and regularization search space for fair comparison. For reference, published results for baselines are: MobileNetV1 (1.0x): 70.6% [15]. MobileNetV2 (1.0x): 72.0% [36]. MnasNet-A1: 75.2% [41]. ResNet-50: 76.4% [11]. EfficientNet-B0: 76.3% [42].

Table 2: COCO object detection minival performance of our CondConv-MobileNetV1 SSD 300 architecture with 8 experts per layer. Mean average precision (mAP) reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95). CondConv improves mAP at all model sizes with small relative increase in inference cost (<5%).

	Baseline		CondConv	
	MADDs ($\times 10^6$)	mAP	MADDs ($\times 10^6$)	mAP
MobileNetV1 (0.5x)	352	14.4	363	18.0
MobileNetV1 (0.75x)	730	18.2	755	21.0
MobileNetV1(1.0x)	1230	20.3	1280	22.4

Table 3: Different routing architectures. Our baseline CondConv(CC)-MobileNetV1 uses a one-layer, fully-connected routing function with Sigmoid activation for each CondConv block.

Routing Fn	MADDs ($\times 10^6$)	Valid Top-1 (%)
CC-MobileNetV1 (0.25x)	55.7	62.0
Single	55.5	56.5
Partially-shared	55.6	62.5
Hidden (small)	55.6	57.7
Hidden (medium)	55.9	62.2
Hidden (large)	57.8	54.1
Hierarchical	55.7	60.3
Softmax	55.7	60.5

Table 4: CondConv at different layers in our CondConv(CC)-MobileNetV1 (0.25x) model. FC refers to the final classification layer. CondConv improves performance at every layer.

CondConv Begin Layer	MADDs ($\times 10^6$)	Valid Top-1 (%)
CC-MobileNetV1 (0.25x)	55.7	62.0
MobileNetV1 (0.25x)	41.2	50.0
1	56.3	62.5
5	56.0	62.0
7	55.7	62.0
13	52.5	59.5
15 (FC Only)	49.3	54.2
7 (No FC)	47.6	60.2

of 1024 for 20,000 steps. For the learning rate, we use linear warmup from 0.3 to 0.9 over 1,000 steps, followed by cosine decay [27] from 0.9. We use the data augmentation scheme proposed by Liu et al. [26]. We use the same convolutional feature layer dimensions, SSD hyperparameters, and training hyperparameters across all models. We measure performance as COCO minival mean average precision (mAP) relative to computational cost in multiply-adds (MADDs).

We use our CondConv-MobileNetV1 models with depth multipliers {0.50, 0.75, 1.0} as the feature extractors for object detection. We further replace the additional convolutional feature extractor layers in SSD with CondConv layers.

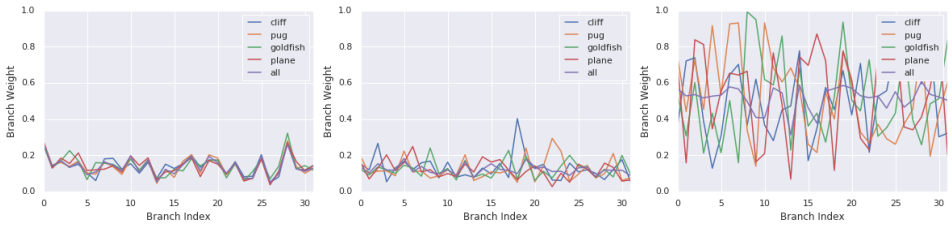
CondConv with 8 experts improves object detection performance at all model sizes (Table 2). Our CondConv-MobileNetV1(0.75x) SSD model exceeds the MobileNetV1(1.0x) SSD baseline by 0.7 mAP at 60% of the inference cost. Moreover, our CondConv-MobileNetV1(1.0x) SSD model improves upon the MobileNetV1(1.0x) SSD baseline by 2.1 mAP at similar inference cost.

4.3 Ablation studies

We perform ablation experiments to better understand model design with the CondConv block. In all experiments, we compare against the same baseline CondConv-MobileNetV1 (0.25x) model with 32 experts per CondConv layer, trained with the same setup as Section 4 and no additional Dropout or data augmentation. The baseline model achieves 61.98% ImageNet Top-1 validation accuracy with 55.7M multiply-adds. The MobileNetV1(0.25x) architecture achieves 50.4% Top-1 accuracy with 41.2M multiply-adds.[4] We choose this setup for ease of training and large effect of CondConv.

[3]Our re-implementation of the baseline models and our CondConv models use the same hyperparameters for fair comparison. As published reference, Howard et al. [15] report mAP of 19.3 for MobileNetV1 (1.0x).

[4]Our implementation. Howard et al. [15] report a top-1 accuracy of 50.0% with different hyperparameters.



(a) Layer 12 (b) Layer 26 (c) Fully Connected (FC)

Figure 3: Mean routing weights for four classes averaged across the ImageNet validation set at three different depths in our CondConv-MobileNetV1 (0.5x) model. CondConv routing weights are more class-specific at greater depth.

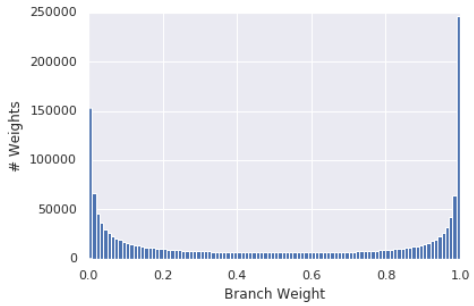


Figure 4: Distribution of routing weights in the final CondConv layer of our CondConv-MobileNetV1 (0.5x) model when evaluated on all images in the ImageNet validation set. Routing weights follow a bi-modal distribution.

4.3.1 Routing function

We investigate different choices for the routing function in Table 3. The baseline model computes new routing weights for each layer. *Single* computes the routing weights only once at CondConv 7 (the 7th separable convolutional block), and uses the same routing weights in all subsequent layers. *Partially-shared* shares the routing weights between every other layer. Both the baseline model and *Partially-shared* significantly outperform *Single*, which suggests that routing at multiple depths in the network improve quality. *Partially-shared* performs slightly outperforms the baseline, suggesting that sharing routing functions among nearby layers can improve quality.

We then experiment with more complex routing functions, by introducing a hidden layer with ReLU activation after the global average pooling step. We vary the hidden layer size to be $input\_dim/8$ for *Hidden (small)*, $input\_dim$ for *Hidden (medium)*, and $input\_dim \cdot 8$ for *Hidden (large)*. Adding a non-linear hidden layer of appropriate size can slightly improve performance. Large hidden layer sizes are prone to over-fitting, even with the same number of experts.

Next, we experiment with *Hierarchical* routing functions, by concatenating the routing weights of the previous layer to the output of the global average pooling layer in the routing function. This adds a dependency between CondConv routing weights, which we find is also prone to overfitting.

Finally, we experiment with the *Softmax* activation function to compute routing weights. The baseline's *Sigmoid* significantly outperforms *Softmax*, which suggests that multiple experts are often useful for a single example.

4.3.2 CondConv Layer Depth

We analyze the effect of CondConv layers at different depths in the CondConv-MobileNetV1 (0.25x) model (Table 4). We use CondConv layers in the begin layer, and all subsequent layers. We further perform ablation studies specific to the final fully-connected classification layer. We find CondConv layers improve performance when applied at every layer in the network. Additionally, we find that additionally applying CondConv before layer 7 in the network has only small effects on performance.

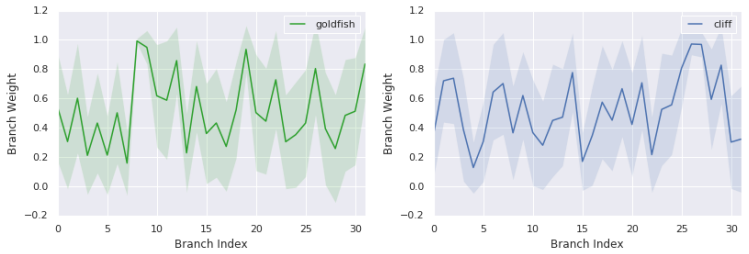


Figure 5: Routing weights in the final CondConv layer in our CondConv-MobileNetV1 (0.5x) model for 2 classes averaged across the ImageNet validation set. Error bars indicate one standard deviation.

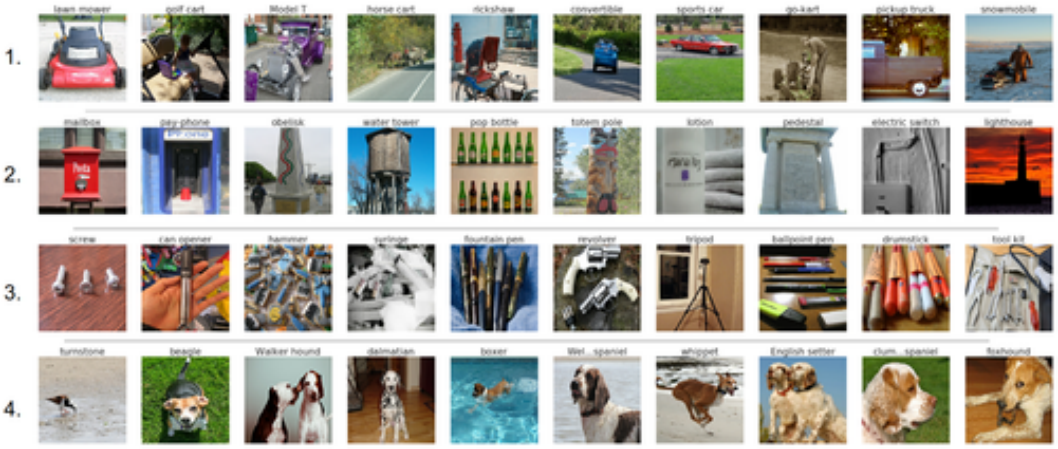


Figure 6: Top 10 classes with highest mean routing weight for 4 different experts in the final CondConv layer in our CondConv-MobileNetV1 (0.5x) model, as measured across the ImageNet validation set. Expert 1 is most activated for wheeled vehicles; expert 2 is most activated for rectangular structures; expert 3 is most activated for cylindrical household objects; expert 4 is most activated for brown and black dog breeds.

For image classification with the CondConv-MobileNetV1 (0.25x) model, CondConv in the final classification layer accounts for a significant fraction of the additional inference cost. Using a normal final classification layer results in smaller performance gains, but is more efficient.

5 Analysis

In this section, we aim to gain a better understanding of the learned kernels and routing functions in our CondConv-MobileNetV1 architecture. We study our CondConv-MobileNetV1 (0.50x) architecture with 32 experts per layer trained on ImageNet with Mixup and AutoAugment, which achieves 71.6% top-1 validation accuracy. We evaluate our CondConv-MobileNetV1 (0.5x) model on the 50,000 ImageNet validation examples, and compute the routing-weights at CondConv layers in the network.

We first study inter-class variation between the routing weights at different layers in the network. We visualize the average routing weight for four different classes (cliff, pug, goldfish, and plane, as suggested by Hu et al. [16] for semantic and appearance diversity) at three different depths in the network (Layer 12, Layer 26, and the final fully-connected layer). The distribution of the routing weights is very similar across classes at early layers in the network, and become more and more class specific at later layers (Figure 3). This suggests an explanation for why replacing additional convolutional layers with CondConv layers near the input of the network does not significantly improve performance.

We next analyze the distribution of the routing weights of the final fully-connected layer in Figure 4. The routing weights follow a bi-modal distribution, with most experts receiving a routing weight

close to 0 or 1. This suggests that the experts are sparsely activated, even without regularization, and further suggests the specialization of the experts.

We then study intra-class variation between the routing weights in the final CondConv layer (Figure 5). Within one class, some kernels are activated with high weight and small variance for all examples. However, even within one class, there can be big variation in the routing weights between examples.

Finally, to better understand experts in the final CondConv layer, we visualize top 10 classes with highest mean routing weight for four difference experts on the ImageNet validation set (Figure 6). We show the exemplar image with highest routing weight within each class. CondConv layers learn to specialize in semantically and visually meaningful ways.

6 Conclusion

In this paper, we proposed conditionally parameterized convolutions (CondConv). CondConv challenges the assumption that convolutional kernels should be shared across all input examples. This introduces a new direction for increasing model capacity while maintaining efficient inference: increase the size and complexity of the kernel-generating function. Since the kernel is computed only once, then convolved across the input, increasing the complexity of the kernel-generating function can be much more efficient than adding additional convolutions or expanding existing convolutions. CondConv also highlights an important research question in the trend towards larger datasets on how to best uncover, represent, and leverage the relationship between examples to improve model performance. In the future, we hope to further explore the design space and limitations of CondConv with larger datasets, more complex kernel-generating functions, and architecture search to design better base architectures.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[2] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network. *arXiv preprint arXiv:1811.11205*, 2018.

[5] Kyunghyun Cho and Yoshua Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*, 2014.

[6] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[7] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[8] Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013.

[9] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[10] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[11] 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.

[12] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard mixtures of experts for large scale weakly supervised vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6865–6873, 2017.

[13] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[15] 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.

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[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:1808.07233*, 2018.

[18] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

[19] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[20] Cem Keskin and Shahram Izadi. Splinenets: Continuous neural decision graphs. In *Advances in Neural Processing Systems*, 2018.

[21] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[23] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404, 1990.

[24] 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 *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[25] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[27] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. 2016.

[28] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Emperical Methods in Natural Language Processing*, 2015.

[29] 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. *arXiv preprint arXiv:1805.00932*, 2018.

[30] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, 2017.

[31] Ravi Teja Mullapudi, William R.Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[32] Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 425–435, 2018.

[33] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[34] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *International Conference on Learning Representations*, 2018.

[35] 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. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[37] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[38] N. Silberman and S. Guadarrama. Tensorflow-slim image classification model library, 2016.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

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

[42] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[43] Mingxing Tan and Quoc V Le. Mixnet: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595*, 2019.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[45] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision*, pages 420–436. Springer, 2018.

[46] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*, 2019.

[47] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[48] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[49] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2017.

Appendices

A ImageNet Architectures

In this section, we provide detailed descriptions of the specific CondConv architectures we used for each baseline model, and elaborate on individual results. All CondConv results are reported with 8 experts per layer. For fair comparison, all results reported use the same training hyperparameters and regularization search space as the CondConv model they are compared against.

CondConv-MobileNetV1. We replace the convolutional layers starting from the sixth separable convolutional block and the final fully-connected classification layer of the baseline MobileNetV1 model with CondConv layers. We share routing weights between depthwise and pointwise layers with a separable convolution block. Our CondConv-MobileNetV1 (0.5x) model with 32 experts per CondConv layer achieves 71.6% accuracy at 190M multiply-adds, comparable to the MobileNetV1 (1.0x) model at 71.7% at 571M multiply-adds.

CondConv-MobileNetV2. We replace the convolutional layers in the final 6 inverted residual blocks and the final fully-connected classification layer of the baseline MobileNetV2 architecture with CondConv layers. We share routing weights between convolutional layers in each inverted bottleneck block. Our CondConv-MobileNetV2 (1.0x) model achieves 74.6% accuracy at 329M multiply-adds. The MobileNetV2 (1.4x) architecture with static convolutions scaled by width multiplier achieves similar accuracy of 74.5% in our implementation (74.7% in [36]), but requires 585M multiply-adds.

CondConv-MnasNet-A1. We replace the convolutional layers in the final 3 block groups of the baseline MnasNet-A1 architecture with CondConv layers. We share routing weights between convolutional layers in each inverted bottleneck block within a block group. The baseline MnasNet-A1 model achieves 74.9% accuracy with 312M multiply-adds. Our CondConv-MnasNet-A1 model achieves 76.2% accuracy with 329M multiply-adds. A larger model from the same search space using static convolutional layers, MnasNet-A2, achieves 75.6% accuracy with 340M multiply-adds [41].

CondConv-ResNet-50. We replace the convolutional layers in the final 3 residual blocks and the final fully-connected classification layer of the baseline ResNet-50 architecture with CondConv layers. The baseline ResNet-50 model achieves 77.7% accuracy at 4096M multiply-adds. Our CondConv-ResNet-50 architecture achieves 78.6% accuracy at 4213 multiply-adds. With sufficient regularization, CondConv improves the performance of even large model architectures with ordinary convolutions that are not optimized for inference time.

CondConv-EfficientNet-B0. We replace the convolutional layers in the final 3 block groups of the baseline EfficientNet-B0 architecture with CondConv layers. We share routing weights between convolutional layers in each inverted bottleneck block within a block group. The baseline EfficientNet-B0 model achieves 77.2% accuracy with 391M multiply-adds. Our CondConv-EfficientNet-B0 model achieves 78.3% accuracy with 413M multiply-adds.

To directly compare our CondConv scaling approach to the compound scaling coefficient proposed by Tan et al. [42], we additionally scale the CondConv-EfficientNet-B0 model with a depth multiplier of 1.1x, which we call CondConv-EfficientNet-B0-depth. Our CondConv-EfficientNet-B0-depth model achieves 79.5% accuracy with only 614M multiply-adds. When trained with the same hyperparameters and regularization search space, the EfficientNet-B1 model, which is scaled from the EfficientNet-B0 model using the compound coefficient, achieves 79.2% accuracy with 700M multiply-adds. In this regime, CondConv scaling outperforms static convolutional scaling with the compound coefficient.