Understanding the Impact of Dynamic Channel Pruning on Conditionally Parameterized Convolutions

Ravi Raju*†
University of Wisconsin-Madison
rraju2@wisc.edu

Dibakar Gope*
Arm ML Research
dibakar.gope@arm.com

Urmish Thakker
Arm ML Research
uthakker@cs.wisc.edu

Jesse Beu
Arm ML Research
jesse.beu@arm.com

ABSTRACT

Recent trends have shown that deep learning models have become larger and more accurate at an increased computational cost, making them difficult to deploy for latency-constrained applications. Conditional execution methods address this increase in cost while preserving the accuracy of the model by only performing the most important computation on important features. In this paper, we analyze a recent method, Feature Boosting and Suppression (FBS), which dynamically assesses which channels contain the most important input-dependent features and prune the others based on a runtime threshold gating mechanism. FBS is able to dynamically prune convolution filters with little loss in accuracy at conservative pruning rates. However, at aggressive pruning rates FBS suffers from heavy accuracy loss in a similar way to aggressive static pruning, due to a low number of active filters and correspondingly narrower effective network per inference. Conditionally parameterized convolutions (CondConv) is another work in the conditional execution domain that increases performance at small computation cost through the generation of input-dependent expert filters. We discover that substituting standard convolutional filters with input-specific filters, as described in CondConv, we enable FBS to address this accuracy loss because CondConv expert filters are able to bolster the narrower network's reduced capacity and capture distinct features from various classes in a single composite filter as needed, making aggressive pruning appealing again. We test our FBS-pruned CondConv model on CIFAR-10 with a custom 9-layer CNN and demonstrate that we can achieve up to 47.2% savings in computational costs at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over the state-of-art FBS technique.

*Both authors contributed equally to this research.
†Work done while interning at Arm ML Research

AIChallengeIoT '20, November 16–19, 2020, Virtual Event, Japan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8134-5/20/11…$15.00
https://doi.org/10.1145/3417313.3429381

CCS CONCEPTS

• **Computing methodologies → Neural networks**.

KEYWORDS

Dynamic Neural Networks, Conditionally Parameterized Convolutions, Dynamic Execution

ACM Reference Format:
Ravi Raju, Dibakar Gope, Urmish Thakker, and Jesse Beu. 2020. Understanding the Impact of Dynamic Channel Pruning on Conditionally Parameterized Convolutions. In *The 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3417313.3429381

1 INTRODUCTION

In recent years, deep neural networks have surpassed human performance on image classification tasks and and speech recognition. The accuracy performance gains are attributed to increasing the model size by adding more layers, which comes at increased computation cost. While these deep models are highly accurate at classification, deploying these models in latency-critical applications, such as self-driving cars and server side real-time video processing, is impractical [34]. These particular applications are unique in that they are not constrained by parameter count, but have strict latency requirements at inference, which motivates the need for a new class of models.

Conditional execution is a paradigm that addresses the computational costs of these models while still preserving their performance. As opposed to more traditional pruning methods which leverage weight sparsity to reduce computation, conditional execution methods preserve the full capacity of a model and accuracy while cutting down computation by selectively performing operations based on an importance criterion. While conditional execution methods do not reduce static model size, they can significantly reduce the number of memory accesses for a single image inference; prior work shows that the peak memory usage is lower than models without conditional execution, which improves cache utilization [6].

There are many flavors of conditional execution but one prevailing viewpoint states that there is unnecessary computation performed for every example; some specific subset of filters will be more activated for a particular class label than another. Feature Boosting and Suppression (FBS) is a state-of-the-art conditional execution mechnism which adopts this viewpoint by dynamically pruning channels on a layer-by-layer basis during inference [6]. FBS introduces auxiliary connections to the convolutional layers which use features from the previous layers to make a decision on which channels should be zeroed out. By training this predictor, FBS adaptively learns which filters to activate or suppress for specific input features. At low to moderate pruning rates, FBS dynamically prunes convolutional filters with little loss in accuracy. However, with a more aggressive pruning rate FBS suffers from heavy accuracy loss in a similar way to aggressive static pruning, due to a low number of active filters and correspondingly narrower effective network per inference, making the classification task more challenging.

Another type of conditional execution is dynamic filter generation which borrows concepts from mixture of experts [26]. Recent work in Conditionally Parameterized Convolutions (CondConv) takes this view in that conventional convolutional weights are replaced with specialized, input-specific filters dynamically created through a linear combination of n experts $(\alpha_1 W_1 + \cdots + \alpha_n W_n)$, where α's are a function of the input [34]. It employs a routing function similar to FBS to encode the important features into the α parameters. CondConv allows for the model capacity to be increased while maintaining a relatively static compute cost since the experts are combined prior to multiplication against the input features when compared to increasing the depth or width of the network.

While analyzing these two methods, we observed a potential opportunity to address the accuracy loss of the state-of-the-art FBS mechanism at high pruning rates. We hypothesize that replacing the conventional convolutional filters in FBS with the high-capacity input-specialized filters of CondConv will directly address the accuracy loss that FBS suffers because of the additional network capacity multiple experts provides. This combined with the ability to create highly efficient composite filters from the combination of experts (which learn distinct features from different classes) counteracts the dynamically lower active filter count and inference-time network narrowing effects of aggressive dynamic pruning. FBS applied to a CondConv model in the experimental results corroborates this hypothesis.

In this paper, our key contribution is that we discover the opportunity to exploit one prominent conditional execution technique, which replaces existing convolutional kernels with high-capacity input-specific kernels [34] for enabling high pruning ability of another state-of-the-art dynamic execution mechanism [6] without compromising model accuracy. Our intuition is that with higher quality expert filters generated with CondConv, FBS can be more aggressive in pruning since fewer expert filters can accomplish the same task as numerous generic filters. We test our FBS-pruned CondConv model on CIFAR-10 with a custom 9-layer CNN and demonstrate that we can achieve up to 47.2% savings in computational costs at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over the state-of-art FBS technique.

2 RELATED WORK

Designing efficient CNNs has been an active, ongoing area of research. In recent years, numerous research efforts have been devoted to compressing neural networks through the use of model pruning [10], quantization [7–9], low-rank tensor decomposition [28–31], compact network architecture design [32], etc.

2.1 Compact Network Architectures

One such approach is creating new custom network architectures. SqueezeNet reduces the number of parameters by replacing them with 1×1 convolutions in the fire module [18]. The MobileNet series introduce multiple changes, such as decomposition of 3×3 convolutions into depthwise convolution and pointwise convolution, adding inverted residuals and linear bottlenecks, and squeeze-excitation layers [14–16, 25]. Shufflenet introduces a pointwise group convolution and channel shuffle operation to reduce operations while preserving accuracy [35]. These architectural optimizations are complementary when compared to our proposal in this work. They can be used in conjunction with our proposal to further reduce model size and computational complexity.

2.2 Sparsity

Several other works deal with the computational efficiency issue by inducing sparsity in the network. Traditional methods involve weight pruning, which zeroes out individual weight parameters which contribute little to the output [11]. Tensor factorization is another model compression approach that decomposes a single layer into multiple small layers [20, 36]. Channel pruning involves trying to prune entire neurons from the network [13, 23]. The key drawback with these works is that the model capacity gets permanently reduced, potentially incurring significant loss in accuracy.

2.3 Conditional Execution

Conditional execution is another approach to efficient deep learning inference. These methods rely on skipping part of an existing model based on the input features [2, 3, 6, 17, 27, 34, 37].

Feature boosting and suppression (FBS) is a state-of-the-art dynamic execution mechanism that dynamically prunes intermediate channels based on input features [6]. It introduces an auxiliary connection, called the channel saliency predictor, to evaluate which channels have the important information content. Based on the output of this predictor, a hyperparameter known as the gate density, d, selects which channels should be pruned in each layer via a top-k winner take all activation function. Once the output of the predictor is obtained, only the convolutional weights corresponding to the surviving channels are used for computation. FBS is preferable to previous static pruning methods because it reduces the dynamic model footprint and minimizes the impact on accuracy while still preserving all neurons in the model.

Conditionally Parameterized Convolutions (CondConv) is another prominent conditional execution technique that replaces conventional convolutional layer with specialized convolutional kernels for each example [34]. In CondConv, the kernels are generated as a linear combination of experts, which are input-dependent scalars. These experts are created in a similar fashion as the auxiliary path in FBS, with the exception there is no pruning of channels. The intuition is rather than increasing the size of the convolution filter, increasing the number of experts in a CondConv layer is more computationally efficient while increasing the model capacity and improving accuracy.

Besides the above two, channel gating neural networks identify regions in the features that contribute less to the output and skips the computation for those specific regions [17]. Precision Gating is a quantization technique that computes most features using low precision and only a small subset of important features with higher precision to preserve accuracy [37]. Batch-shaped channel gated networks use the concept called batch-shaping, matching the marginal aggregate posterior of features in a network to a pre-specified prior distribution, to force the gating mechanism to be more conditional on data [2]. Dynamic Convolution augments the convolution kernels by dynamically scaling them based on their attention, making them input-dependent [3].

3 FEATURE BOOSTING AND SUPPRESSION OF CONDITIONALLY PARAMETERIZED CONVOLUTIONS

While FBS is able to prune filters effectively, one limiting factor of FBS is the requirement of certain minimum number of active filters in all layers to ensure no or minimal degradation in accuracy. This minimum amount of filters places a restriction on how aggressive the pruning rate can be set. However, our observations indicate that we can address this limitation by replacing conventional convolutional kernels with CondConv kernels so that we can prune more heavily with no loss in accuracy.

3.1 Motivation

At moderate pruning rates where a majority of filters are retained, FBS can perform well while reaping substantial gains in computational savings. However, this is not necessarily the case at higher pruning rates since FBS may not retain enough input-specific specialized filters to classify an image correctly. Because FBS does not have as much representational ability at dynamically-narrow aggressive pruning rates, it becomes more challenging to differentiate between classes. Despite this, there is strong motivation to prune as aggressively as possible to capture the benefits of a quadratic savings in compute.

CondConv can resolve some of the issues FBS suffers through the generation of input-specific filters since CondConv may select and combine from a varied set of experts for distinct classes. By separating which experts get activated for a particular class, we may more easily discriminate between which filters are candidates to be pruned since the probability of two different classes sharing input-specific filters is low. In other words, CondConv may enable FBS to aggressively prune a majority of the other input-specific filters since a smaller number of these filters may be sufficient to classify the input when compared to the baseline FBS, which needs to prune generic (non-input specific) filters prone to destructive interference.

To illustrate the points presented above, consider an example layer in a FBS network layer with 32 channels for a toy problem with 10 classes. If the pruning rate (gate density in this case) is set 0.5, half of the filters are eliminated per-inference, but classification accuracy may be sustained since the remaining 16 filters are still sufficient to classify the input. If the pruning rate is set too aggressively though, e.g. to 0.3, it is plausible that using only 10 filters does not have enough representative power to distinguish between classes in all cases and would cause a drop in accuracy. However, if the generic filters in this FBS layer are replaced with ConvConv filters with 4 experts, two important observations arise: (1) the model capacity is inflated by 4×, granting much more representational ability to the layer, and (2) since the CondConv filters are created in a fine-grained, input-specific manner, good classification may not require all the experts. These reasons are an explanation for why using CondConv filters may allow FBS to use a high pruning rate with little to no loss in accuracy.

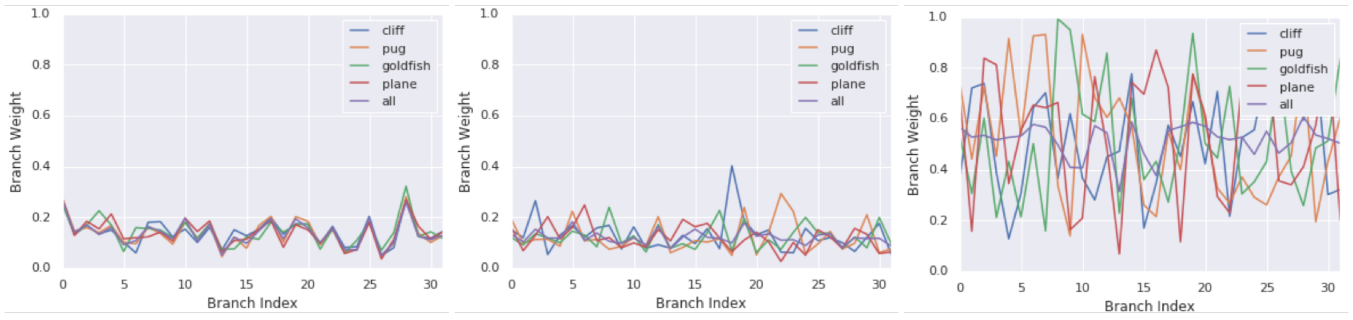
To reiterate, CondConv experts learn distinct features from certain classes and not others. Figure 1, taken from [34], plots the mean routing weights for MobileNetV1 with 32 experts on 4 classes of the ImageNet validation set [4]. For earlier layers in the network, as in 1a, experts are not learning specific filters which differentiate between classes, implying generally the dominance of useful low-level features. This figure shows that replacing the convolution layer with a CondConv layer is akin to increasing the depth of the layer. From Figure 1b we see deeper layer experts become more input specific and certain combinations of experts become active for distinct classes. At the penultimate layer in Figure 1c, only one or two experts are active for each class example indicating that in the deepest layers of the network CondConv kernels resemble a mixture of experts. Although not as pronounced, specialization of filters is present at layer 26 as shown in Figure 1b. However, filters become more individual and specialized deeper in the network as demonstrated in Figure 1c. The authors of [34] further corroborates this observation that CondConv experts specialize, going so far as to graphically demonstrate a specific instance of a filter whose experts specializes in wheeled vehicles, rectangles, cylinders and dogs. The key takeaway is that which experts even get activated by the auxiliary paths or routing function preemptively reveals which class the input is likely to belong to and thus reduces the scope of the classification task, which allows for more pruning opportunity.

This key insight motivates applying dynamic channel pruning, similar to FBS, to a model made of input-specific expert filters, as in CondConv, in order to reap more computational savings while ensuring little to no loss in model accuracy. Experts becoming highly specialized imply that fewer filters are needed for proper prediction. With higher quality filters, FBS can take advantage of the other irrelevant, specialized filters and prune them at an aggressive rate with no accuracy penalty. In the following subsections, we provide detailed descriptions of expert filter generation in CondConv and then show how FBS can be applied to enable abundant pruning of expert filters, resulting in significant reductions in computational cost.

3.2 Designing a Conditionally Parameterized Convolutional Layer

We consider a CNN, $F(\cdot)$, with L layers with the following definition, $y = F(x_{in}) = f_L(\cdots f_2(f_1(x_0))\cdots)$, in which the n^{th} layer $f_n : \mathbb{R}^{C_{n-1}\times H_{n-1}\times W_{n-1}} \longrightarrow \mathbb{R}^{C_n\times H_n\times W_n}$ and x_{in}, y are the input and output respectively. Each layer in the network will produce an output with C_n channels of features with height H_n and width W_n. Each f is described by a convolution operation, followed by an activation function. The n^{th} layer can be expressed as

$$f_n(x_{n-1}) = \sigma_n(\gamma_n \cdot norm(conv_n(x_{n-1}, W_n)) + \beta_n). \tag{1}$$



(a) Layer 12 **(b) Layer** 26 **(c) Full Connected (FC)**

Figure 1: Mean CondConv routing weights for four classes averaged across the ImageNet validation set at three different depths (a) layer 12**, (b) layer** 26 **and (c) the FC layer of MobileNetV1. CondConv routing weights become more class-specific at deep layers. Figure is from [34].**

In the above equation, $conv_n$ is parameterized by weight tensors, $W_n \in \mathbb{R}^{C^l \times C^{l-1} \times k \times k}$ (k being the kernel size), and input from the previous layer $x_{n-1} \in \mathbb{R}^{C^{l-1} \times H^{l-1} \times W^{l-1}}$. $norm$ is the batch normalization operation [19] and σ_n denotes the activation function at the n^{th} layer. γ_n and β_n are introduced as trainable parameters for every convolutional layer which will be introduced in section 3.3.

Adding CondConv into the derivation above is simple in that only the $conv_n$ operation gets replaced. The input-specific CondConv filters are generated with a routing function. The concrete definition of CondConv at layer n takes the following form:

$$conv_{cond}(x, W) = (\alpha_1 W_1 + \cdots + \alpha_n W_n) * x$$

$$f_n(x_{n-1}) = \sigma_n(\gamma_n \cdot norm(conv_{cond}(x_{n-1}, W_n)) + \beta_n), \tag{2}$$

where α values represent the expert scalar weights, n is the number of experts, and σ is the activation function. Note that each weight W in the CondConv operation has the same dimension as the weight in the original layer. While the capacity of each layer is increased, CondConv still incurs computational costs on par with conventional convolution operation since it performs only one convolution after linearly combining the experts. The small overhead to pay is the generation of routing weights and the linear combination of n experts using them before applying the convolution.

The routing function is an integral component for CondConv to generate input-specific filters while still being computationally efficient. It is comprised of steps, such as Global Average Pooling (GAP) operation, fully-connected (FC) layer operation, and sigmoid activation function. The GAP step reduces the dimension of the previous layer input into a 1-d vector to reduce computation. Assuming each filter has n experts, the FC layer, initialized with weight $R \in \mathbb{R}^{C_{in}} \longrightarrow \mathbb{R}^n$, takes in the vector to generate n scalar values, subsequently passed through a sigmoid activation function to produce routing weights. The sigmoid activation function is chosen for this case, since empirically, it has better performance than other choices such as softmax, etc. The routing function design is detailed below with x being the previous layer input.

$$\alpha = Sigmoid(GAP(x)R). \tag{3}$$

It is possible to use more complex routing functions like multi-layer perceptrons (MLPs) to potentially obtain better expert weights at the cost of increased computation. However, these complex functions tend to be prone to overfit so we stick to using a single FC layer. With this in-depth understanding of creating CondConv filters, in the next subsection, we explore the finer details of FBS and how expert filters allow for a higher pruning rate.

3.3 Feature Boosting and Suppression with Channel Saliencies

Recall from Equation 2 that γ_n and β_n are trainable parameters. FBS essentially exploits these trainable parameters to dynamically boost and suppress channels. γ_n can be thought of a batch normalization scale factor and is the mechanism that decides which channels will be zeroed out. It is calculated for layer n as follows:

$$\gamma_n(x_{n-1}) = wta_{\lceil dC_n \rceil}(g_n(x_{n-1})). \tag{4}$$

Here, $wta_k(\cdot)$ is a k-winner take all activation function which selects the top k entries in the input and zeros out the rest. The number of channels that are zeroed out are determined by a hyperparameter known as the gate density ratio, d, which controls the level of pruning. In other words, $wta_{\lceil dC_n \rceil}$ takes the top dC_n channels and only performs the convolution with respect to the surviving channels. It important to note that none of the expert scalar weights, αs, are pruned but rather the input-specific CondConv filters.

FBS has a similar routing mechanism to CondConv called the channel saliency predictor, represented as $g_n(x_{n-1})$ in Equation 2. It is also implemented as a low-cost FC layer appended to the convolutional layers. The predictor takes the previous layer input and calculates the channel importance so the wta activation function can rank the channels accordingly. The predictor maps the previous layer input to a 1-d vector by subsampling to reduce computation cost, $ss : \mathbb{R}^{C_{n-1} \times H \times W} \longrightarrow \mathbb{R}^{C_n}$:

$$ss(x_{n-1}) = \frac{1}{HW}[s(x_{n-1}^1)s(x_{n-1}^2) \cdots s(x_{n-1}^C)]. \tag{5}$$

$s(x_{n-1}^c)$ reduces the c^{th} channel of the input to a scalar through the ℓ_1 norm, $||x_{n-1}||$, for instance which is effectively a GAP operation. Various choices of norms for the subsampling function, such as $||\ell_2||$, ℓ_∞, etc., can be considered but we select the ℓ_1 norm for simplicity. After the subsampling operation, the channel saliency predictor is constructed with a weight tensor, $\phi_n \in \mathbb{R}^{C^n \times C^{n-1}}$ and bias, $\rho_n \in \mathbb{R}^{C^n}$:

$$g_n(x_{n-1}) = \sigma_n(ss(x_{n-1})\phi_n + \rho_n). \tag{6}$$

Just as in CondConv, the routing function can be more complex than a single FC layer but again is prone to overfitting; thus, we choose one single FC layer for the predictor. Since both routing functions use the same input, CondConv will generate filters that will more readily capture the most relevant features and FBS will aid in this process by eliminating the irrelevant channels. This synergy between these methods is the enabling factor for FBS to use a higher pruning rate and win on computational savings with negligible loss in accuracy.

3.4 Training Methodology

From the onset of training, FBS is applied to prune a network with CondConv layers while CondConv layers are simultaneously being trained as opposed to training the CondConv layers in absence of FBS first and fine tuning the resultant network with turning on FBS later. Since the wta activation function is a piecewise and differentiable, standard gradient descent (GD) training methods can be used to train the network. For the loss function, we choose the standard cross entropy loss with L_2 weight decay, $\lambda_{wd} = 10^{-4}$. We also impose a Lasso constraint on the the output of the channel saliency predictor $g_l(x_{l-1}) : \lambda \sum_{l=1}^{n} ||g(x_{l-1})||_1$, where we set $\lambda = 10^{-8}$.

4 EXPERIMENTS AND RESULTS

4.1 Architecture, Datasets and Experimental Setup

We primarily conduct experiments on the CIFAR-10 dataset, which is a standard vision benchmark in deep learning. The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [21].

For the baseline model, we use M-CifarNet, a custom 9-layer CNN. This M-CifarNet model is adopted from the state-of-the-art FBS work [6]. The baseline M-CifarNet model achieves an accuracy of 94.84%. We train all our models in the TensorFlow framework [1]. We replace the convolutional filters in the M-CifarNet model with CondConv filters, adding the appropriate routing functional. In all our experiments, we fix the number of CondConv experts in each layer (n) to 4.

Gate Density (d)	MACs (MOps)	MAC savings
1.0	174.3	1x
0.9	145.5	1.20x
0.7	88.3	1.97x
0.5	45.3	3.85x
0.3	16.5	10.58x
0.25	11.5	15.1x

Table 1: Number of MAC operations and savings at different gate densities (equivalent to (1 - d) pruning rates under SCP)

We train and apply FBS to the baseline model and the model with CondConv filters for 480 epochs. We use standard data augmentation techniques such as random flip, random crop (with a value of 4), data whitening, in addition to cutout regularization, setting the cutout parameter to 18 [5]. We use RMSProp [33] as the optimizer with the learning rate initially set to 0.01 and a cosine annealing decay learning rate scheduler [22]. We set our batch size to 256 for all the networks we trained. We initialize all network parameters with He initialization [12]. We use the swish activation function [24] for all the layers in the model with the exception of the CondConv routing function, which exclusively uses sigmoid. We also applied gradient clipping to ensure smoother convergence during the training process of our FBS-pruned CondConv models. We measure performance as top-1 accuracy on the test set and the computational cost in multiple-accumulate (MACs) operations. In Table 1, we show the MACs operations for FBS-pruned CondConv at various gate densities as well as its associated MAC savings. We omit reporting the MACs and savings for FBS since FBS will incur a similar computation cost to that of the FBS-pruned CondConv since the additional overhead of the routing function computation and the combination of experts is marginal (2.94% at 0.5 gate density over FBS) in comparison to the cost of the convolution operation at different gate densities.

4.2 CIFAR-10 Classification

We demonstrate the accuracy of FBS-pruned CondConv at different gate densities and compare this against the baseline FBS in Figure 2. The y-axis reports the accuracy, while the x-axis reports the number of operations (measured in MACs) at different gate densities. At every gate density, FBS-pruned CondConv clearly outperforms FBS in its ability to retain accuracy at the expense of reduced computation cost. For example, FBS-pruned CondConv at gate density $d = 0.5$ is able to offer similar accuracy to that of baseline FBS at $d = 0.7$ while incurring about 43M fewer MAC operations. Similarly, FBS-pruned CondConv at $d = 0.25$ can match the accuracy of baseline FBS at $d = 0.3$ while incurring about 5M fewer MAC operations. The trend in Figure 2 illustrates that as the pruning rate becomes more aggressive, the performance gap between FBS-pruned CondConv and FBS becomes wider. This result affirms our hypothesis that the input-specific filters from CondConv enable FBS to prune aggressively with negligible accuracy degradation and supports the assertion that a smaller set of expert filters can accomplish the same task as a larger number of generic filters.

In order to validate the potential of FBS-pruned CondConv over statically channel pruned CondConv network (SCP-pruned CondConv), we trained a CondConv model with statically pruning away channels proportional to different gate density ratios of FBS. The accuracy results are shown in Table 2. FBS-pruned CondConv significantly outperforms SCP-pruned CondConv with accuracy margins of 1.17% and 1.36% at the highest reported pruning rates of $d = 0.3$ and $d = 0.25$, respectively. This analysis highlights the advantage that conditional execution techniques such as FBS provides over static pruning methods by maintaining model capacity and minimizing the accuracy penalty.

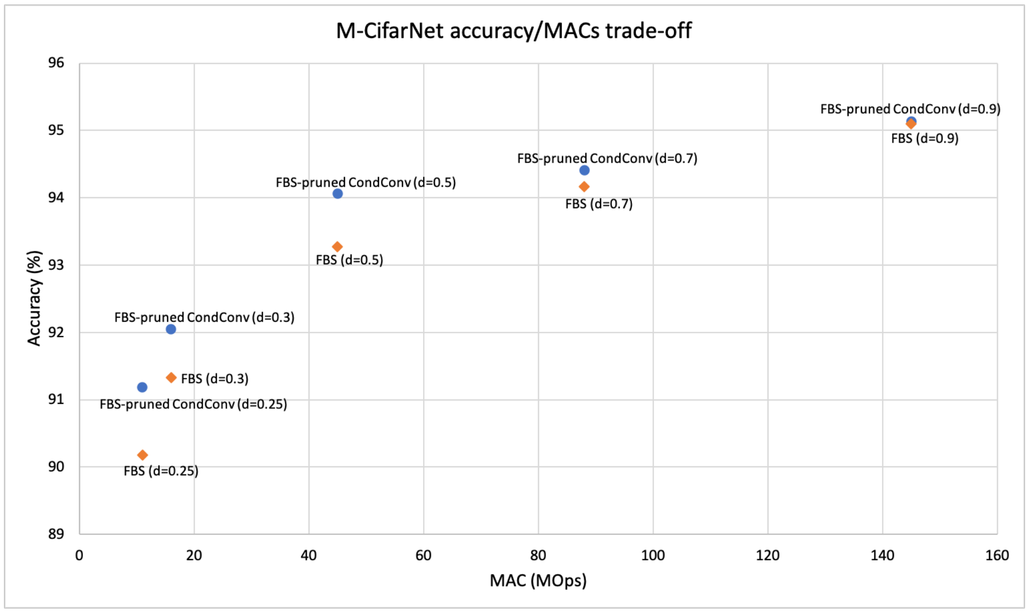


Figure 2: Accuracy/MACs trade-off for baseline FBS and the FBS-pruned CondConv model. For lower MAC operation points, FBS-pruned CondConv's accuracy is significantly higher than that of the FBS baseline. d is the gate density of FBS mechanism.

Gate Density (d)	SCP Accuracy (%)	SCP-pruned CondConv Accuracy (%)	FBS-pruned CondConv Accuracy (%)
0.9	94.91	94.85	95.13
0.7	94.17	94.03	94.41
0.5	93.12	93.43	94.06
0.3	90.32	90.88	92.05
0.25	88.56	89.83	91.19

Table 2: Comparison of FBS-pruned CondConv with SCP-pruned CondConv Accuracy and SCP at different MAC budgets.

5 CONCLUSION

In this work, we demonstrate the synergy between two state-of-the-art conditional execution techniques to achieve aggressive pruning ability of convolutional networks. This in turn results in large computation savings over existing dynamic execution mechanisms while ensuring little to no loss in model accuracy. The input-specific nature of CondConv filters as well as their unique property of separating classes by experts addresses the destructive interference issue that FBS experiences in high pruning regimes. We conducted CIFAR-10 experiments showing FBS-pruned CondConv can achieve up to 47.2% reduction in computational cost at iso-accuracy and 1.01% improvement in accuracy at iso-computational costs over FBS. Note that this work was targeted towards achieving high accuracy while minimizing computational costs albeit at the cost of increasing model size from the use of CondConv kernels. In future, we would like to combine our FBS-pruned CondConv technique with traditional pruning strategies to address the static memory footprint and scale our insights to larger models and ImageNet-like challenging datasets. We would also incorporate other conditional execution techniques into our FBS-pruned CondConv model to push the boundary of the accuracy-MACs tradeoff further. We leave this exploration for future work.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[2] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. 2019. Batch-Shaping for Learning Conditional Channel Gated Networks. arXiv:1907.06627 [cs.LG]

[3] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. 2019. Dynamic Convolution: Attention over Convolution Kernels. arXiv:1912.03458 [cs.CV]

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

[5] Terrance DeVries and Graham W. Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. arXiv:1708.04552 [cs.CV]

[6] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. 2018. Dynamic Channel Pruning: Feature Boosting and Suppression. arXiv:1810.05331 [cs.CV]

[7] Dibakar Gope, Jesse Beu, Urmish Thakker, and Matthew Mattina. 2020. Ternary MobileNets via Per-Layer Hybrid Filter Banks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[8] Dibakar Gope, Jesse G. Beu, Urmish Thakker, and Matthew Mattina. 2020. Aggressive Compression of MobileNets Using Hybrid Ternary Layers. *tinyML Summit* (2020). https://www.tinyml.org/summit/abstracts/Gope_Dibakar_poster_abstract.pdf

[9] Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2019. Ternary Hybrid Neural-Tree Networks for Highly Constrained IoT Applications. In *Proceedings of Machine Learning and Systems 2019*. 190–200.

[10] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). arXiv:1510.00149

[11] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149 [cs.CV]

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. arXiv:1502.01852 [cs.CV]

[13] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. arXiv:1707.06168 [cs.CV]

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. arXiv:1905.02244 [cs.CV]

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs.CV]

[16] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2017. Squeeze-and-Excitation Networks. arXiv:1709.01507 [cs.CV]

[17] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. 2018. Channel Gating Neural Networks. arXiv:1805.12549 [cs.LG]

[18] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360 [cs.CV]

[19] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [cs.LG]

[20] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions. arXiv:1405.3866 [cs.CV]

[21] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n.d.]. CIFAR-10 (Canadian Institute for Advanced Research). ([n. d.]). http://www.cs.toronto.edu/~kriz/cifar.html

[22] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv:1608.03983 [cs.LG]

[23] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning Convolutional Neural Networks for Resource Efficient Inference. arXiv:1611.06440 [cs.LG]

[24] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for Activation Functions. arXiv:1710.05941 [cs.NE]

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs.CV]

[26] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538 [cs.LG]

[27] Jin Tao, Urmish Thakker, Ganesh Dasika, and Jesse Beu. 2019. Skipping RNN State Updates without Retraining the Original Model. In *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems* (New York, NY, USA) *(SenSys-ML 2019)*. Association for Computing Machinery, New York, NY, USA, 31–36. https://doi.org/10.1145/3362743.3362965

[28] Urmish Thakker, Jesse Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2020. Rank and run-time aware compression of NLP Applications. arXiv:2010.03193 [cs.CL]

[29] Urmish Thakker, Jesse G. Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2019. Run-Time Efficient RNN Compression for Inference on Edge Devices. *CoRR* abs/1906.04886 (2019). arXiv:1906.04886

[30] Urmish Thakker, Jesse G. Beu, Dibakar Gope, Chu Zhou, Igor Fedorov, Ganesh Dasika, and Matthew Mattina. 2019. Compressing RNNs for IoT devices by 15-38x using Kronecker Products. *CoRR* abs/1906.02876 (2019). arXiv:1906.02876

[31] Urmish Thakker, Igor Fedorov, Jesse Beu, Dibakar Gope, Chu Zhou, Ganesh Dasika, and Matthew Mattina. 2019. Pushing the limits of RNN Compression. *CoRR* abs/1910.02558 (2019). arXiv:1910.02558

[32] Urmish Thakker, Paul Whatmough, Zhi-Gang Liu, Matthew Mattina, and Jesse Beu. 2020. Compressing Language Models using Doped Kronecker Products. arXiv:2001.08896 [cs.LG]

[33] T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

[34] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. 2019. CondConv: Conditionally Parameterized Convolutions for Efficient Inference. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 1307–1318. http://papers.nips.cc/paper/8412-condconv-conditionally-parameterized-convolutions-for-efficient-inference.pdf

[35] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. arXiv:1707.01083 [cs.CV]

[36] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2015. Accelerating Very Deep Convolutional Networks for Classification and Detection. arXiv:1505.06798 [cs.CV]

[37] Yichi Zhang, Ritchie Zhao, Weizhe Hua, Nayun Xu, G. Edward Suh, and Zhiru Zhang. 2020. Precision Gating: Improving Neural Network Efficiency with Dynamic Dual-Precision Activations. arXiv:2002.07136 [cs.CV]