Dirichlet Pruning for Neural Network Compression

Kamil Adamczewski
Max Planck Institute for Intelligent Systems
ETH Zürich

Mijung Park
Mac Planck Institute for Intelligent Systems
University of Tübingen

Abstract

We introduce *Dirichlet pruning*, a novel post-processing technique to transform a large neural network model into a compressed one. Dirichlet pruning is a form of structured pruning which assigns the Dirichlet distribution over each layer's channels in convolutional layers (or neurons in fully-connected layers), and estimates the parameters of the distribution over these units using variational inference. The learned distribution allows us to remove unimportant units, resulting in a compact architecture containing only crucial features for a task at hand. The number of newly introduced Dirichlet parameters is only linear in the number of channels, which allows for rapid training, requiring as little as one epoch to converge. We perform extensive experiments, in particular on larger architectures such as VGG and ResNet (94% and 72% compression rate, respectively) where our method achieves the state-of-the-art compression performance and provides interpretable features as a by-product.

1 INTRODUCTION

Neural network models have achieved state-of-the art results in various tasks, including object recognition and reinforcement learning [6, 9, 30, 1, 5]. The algorithmic and hardware advances propelled the network sizes which have increased several orders of magnitude, from the LeNet [22] architecture with a few thousand parameters to ResNet [12] architectures with almost 100 million parameters. Recent language models require striking 175 billion parameters [3]. However, large architectures incur high computational costs and memory requirements at both training and test time. They also become hard to analyze and interpret. Besides, it is unclear whether a network needs all the parameters given by a hand-picked, rather than intelligently-designed architecture. For example, VGG-16 [34] consists of layers containing 64, 128, 256, and 512 channels, respectively. However, there is no evidence that all those channels are necessary for maintaining the model's generalization ability.

Previous work noticed and addressed these redundancies in neural network architectures [23, 11]. Subsequently, neural network compression became a popular research topic, proposing smaller, slimmer, and faster networks while maintaining little or no loss in the immense networks' accuracy [15, 16, 18]. However, many of existing approaches judge the importance of weight parameters relying on the proxies such as weights' magnitude in terms of L1 or L2 norms [13]. In this work, we take a different route by learning the importance of a computational unit, a channel in convolutional layers or a neuron in fully connected layers. For simplicity, we will use the term, channels, as removable units throughout the paper, with a focus on convolutional neural networks (CNNs).

Our pruning technique provides a numerical way to compress the network by introducing a new and simple operation per layer to existing neural network architectures. These operations capture the relative importance of each channel to a given task. We remove the channels with low importance to obtain a compact representation of a network as a form of structured pruning.

The learned importance of channels also naturally provides a ranking among the channels in terms of their significance. Visualizing the feature maps associated with highly-ranked channels provides intuition why compression works and what information is encoded in the remaining channels after pruning.

Taken together, we summarize our contributions as follows:

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

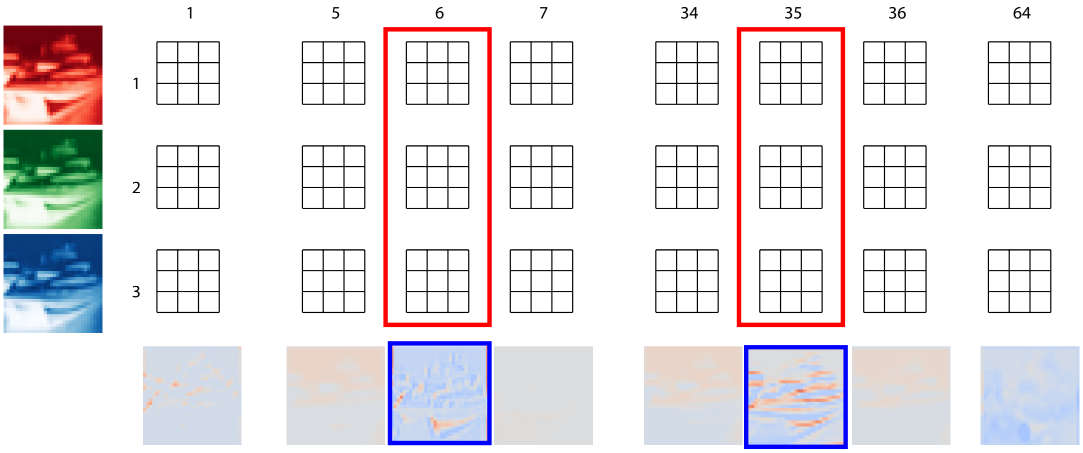


Figure 1: First layer (convolutional layer) of the VGG-16 architecture as an example of parameter layout. In the case of convolutional layer, a *convolutional neuron* is equivalent to a channel, which consists of a set of filters. In the example above, the input contains three channels (R,G,B) and the output contains 64 channels. We name these channels with ordinary numbers from 1 to 64. Due to the space limit, we only show the outputs of channels $1, 5, 6, 7, 34, 35, 36, 64$. In this work, we propose to learn the importance of the (output) channels. The two channels outlined in red are the example channels which scored high in the importance. As the output feature maps show (in the blue boxes), the important channels contain humanly-interpretable visual cues. As in structured pruning, we remove the entire channels of less importance such as 7 and 36, while we keep the informative channels such 6 and 35.

- **A novel pruning technique**. We propose a novel structured pruning technique which learns the importance of the channels for any pre-trained models, providing a practical solution for compressing neural network models. To learn the importance, we introduce an additional, simple operation to the existing neural network architectures, called an *importance switch*. We assigns the Dirichlet distribution over the importance switch, and estimate the parameters of the distribution through variational inference. The learned distribution provides a relative importance of each channel for a task of interest.
- **Speedy learning**. Parameter estimation for the importance switch is fast. One epoch is often enough to converge.
- **Insights on neural network compression.** Our method allows us to rank the channels in terms of their learned importance. Visualizing the feature maps of important channels provides insight into which features are essential to the neural network model's task. This intuition explains why neural network compression works at all.
- **Extensive experiments for compression tasks**. We perform extensive experiments to test our method on various architectures and datasets. By learning which channels are unimportant and pruning them out, our method can effectively compress the networks. Its performance excels across a range of pruning rates.

2 RELATED WORK

The main motivation behind this work is to decrease the size of the network to the set of essential and explainable features, without sacrificing a model's performance. To this end, we slim the network by identifying and removing the redundant channels as a form of structured network pruning [31, 10]. Compared to weight pruning that removes each individual weight, structured pruning [19] that removes channels in convolutional layers or neurons in fully-connected layers, provides practical acceleration.

Most common pruning approaches take into account the magnitude of the weights and remove the parameters with the smallest L1 or L2-norm [10]. Alternatively, gradient information is used to approximate the impact of parameter variation on the loss function [22, 31]. In these works, magnitude or a Hessian, respectively, serve as proxies for parameter importance.

Our work follows the line of research which applies probabilistic thinking to network pruning. A common framework for these methods utilizes Bayesian paradigm and design particular type of priors (e.g. Horseshoe or half-Cauchy prior) which induce sparsity in the network [31, 38, 27, 33]. In our work, we also apply Bayesian formalism, however we do not train the model from scratch using sparse priors. Instead, given any pre-trained model, we learn the importance of the channels and prune out those with less importance, as a post-processing step. We also apply Dirichlet distribution as prior and posterior for learning the channel importance, which has not been seen in the literature.

Many of the Bayesian approaches assign a distribution over the single weight vector, and, in the case of Bayesian neural networks, perform the variational inference using the mean-field approximation for the computational tractability [2], which introduces a large number of parameters, and can be slow or impractical. On the other hand, our approach is practical. It learns the importance of channels as groups of weight vectors, and introduces the number of parameters linear in the number of channels in the network.

One may also find resemblance between the proposed method and attention mechanisms which accentuate certain elements. Dirichlet pruning does something similar, but in a much simpler way. We do not build attention modules (like e.g. [40] which uses neural networks as attention modules), only take a rather simple approach by introducing only the number of Dirichlet parameters equal to the number of channels, and learning them in a Bayesian way.

Dirichlet pruning allows optimizing single layers at a time, or the entire layers simultaneously as in [42]. In some sense, our work adopts certain aspects of dynamic pruning [8] since we automate the neural network architecture design by learning the importance of channels. We perform a short fine-tuning on the remaining channels, resulting in a fast and scalable retraining.

3 METHOD

Given a pre-trained neural network model, our method consists of two steps. In the first step, we freeze the original network's parameters, and only learn the importance of the channels (please refer to Fig. 1 for visual definition). In the second step, we discard the channels with low importance, and fine-tune the original network's parameters. What comes next describes our method in detail.

3.1 Importance switch

To learn the importance of channels in each layer, we propose to make a slight modification in the existing neural network architecture. We introduce a new component, *importance switch*, denoted by $\mathbf{s}_l$ for each layer l. Each importance switch is a probability vector of length D_l, where D_l is the output dimension of the lth fully-connected layer or the number of output channels of the lth layer[1]. As it is a probability vector, we ensure that the sum across the elements of the vector is 1: $\sum_j^{D_l} \mathbf{s}_{l,j} = 1$. The switch $\mathbf{s}_{l,j}$ is the jth element of the vector, corresponding to the jth output channel on the layer, and its value is learned to represent the normalized importance (as the sum of elements is 1) of that channel.

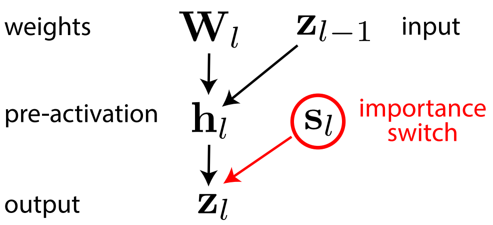


Figure 2: Modification of a neural network architecture by introducing *importance switch* per layer. Typically, an input to the lth layer $\mathbf{z}_{l-1}$ and the weights $\mathbf{W}_l$ defined by channels form a pre-activation, which goes through a nonlinearity σ to produce the layer's output $\mathbf{z}_l = \sigma(\mathbf{h}_l)$. Under our modification, the pre-activation is multiplied by the importance switch then goes through the nonlinearity $\mathbf{z}_l = \sigma(\mathbf{s}_l \circ \mathbf{h}_l)$.

Introducing a switch operation in each layer in a neural network model may bare similarity to [28, 24], where the switch is a binary random variable and hence can only *select* which channels are important. By contrast, our importance switch provides *the degree of importance* of each channel.

With the addition of importance switch, we rewrite the forward pass under a neural network model, where the function $f(\mathbf{W}_l, \mathbf{x}_i)$ can be the convolution operation for convolutional layers, or a simple matrix multiplication between the weights $\mathbf{W}_l$ and the unit $\mathbf{x}_i$ for fully-connected layers, the pre-activation is given by

$$\mathbf{h}_{l,i} = f(\mathbf{W}_l, \mathbf{x}_i), \tag{1}$$

and the input to the next layer after going through a nonlinearity σ, multiplied by a switch $\mathbf{s}_l$, is

$$\mathbf{z}_{l,i} = \sigma(\mathbf{s}_l \circ \mathbf{h}_{l,i}), \tag{2}$$

where $\circ$ is an element-wise product.

The output class probability under such networks with L hidden layers for solving classification problems can

[1] Notice that the number of output channels in the layer l is the same as the number of input channels in the layer l+1. Importance switch vector S_l is defined over the output channels. However, pruning layer l's output channels also reduces the number of input channels in the layer $l + 1$.

be written as

$$P(\mathbf{y}_i|\mathbf{x}_i, \{\mathbf{W}_l\}_{l=1}^{L+1}) = g\left(\mathbf{W}_{L+1}\mathbf{z}_{L,i}\right), \qquad (3)$$

where $\mathbf{z}_{L,i} = \sigma(\mathbf{s}_L \circ [f(\mathbf{W}_L \mathbf{z}_{L-1,i})])$ and g is e.g. the *softmax* operation. A schematic of one-layer propagation of the input with the importance switch is given in Fig. 2.

3.2 Prior over importance switch

We impose a prior distribution over the importance switch using the Dirichlet distribution with parameters $\boldsymbol{\alpha}_0$:

$$p(\mathbf{s}_l) = \mathrm{Dir}(\mathbf{s}_l; \boldsymbol{\alpha}_0). \qquad (4)$$

Our choice for the Dirichlet distribution is deliberate: as a sample from this Dirichlet distribution sums to 1, each element of the sample can encode the importance of each channel in that layer.

As we typically do not have prior knowledge on which channels would be more important for the network's output, we treat them all equally important features by setting the same value to each parameter, i.e., $\boldsymbol{\alpha}_0 = \alpha_0 * \mathbf{1}_{D_l}$ where $\mathbf{1}_{D_l}$ is a vector of ones of length D_l [2]. When we apply the same parameter to each dimension, this special case of Dirichlet distribution is called *symmetric* Dirichlet distribution. In this case, if we set $\alpha_0 < 1$, this puts the probability mass toward a few components, resulting in only a few components that are non-zero, i.e., inducing sparse probability vector. If we set $\alpha_0 > 1$, all components become similar to each other. Apart from the flexibility of varying α, the advantage of Dirichlet probability distribution is that it allows to learn the relative importance which is our objective in creating a ranking of the channels.

3.3 Posterior over importance switch

We model the posterior over $\mathbf{s}_l$ as the Dirichlet distribution as well but with *asymmetric* form to learn a different probability on different elements of the switch (or channels), using a set of parameters (the parameters for the posterior). We denote the parameters by $\boldsymbol{\phi}_l$, where each element of the vector can choose any values greater than 0. Our posterior distribution over the importance switch is defined by

$$q(\mathbf{s}_l) = \mathrm{Dir}(\mathbf{s}_l; \boldsymbol{\phi}_l). \qquad (5)$$

3.4 Variational learning of importance switches

Having introduced the formulation of importance switch, we subsequently proceed to describe how to estimate the distribution for the importance switch. Given the data $\mathcal{D}$ and the prior distribution over the importance switch $p(\mathbf{s}_l)$ given in eq. 4, we shall search for the posterior distribution, $p(\mathbf{s}_l|\mathcal{D})$. Exact posterior inference under neural network models is not analytically tractable. Instead, we resort to the family of *variational* algorithms which attempt to optimize the original distribution $p(\mathbf{s}_l|\mathcal{D})$ with an approximate distribution $q(\mathbf{s})$ by means of minimizing the Kullback-Leibler (KL) divergence:

$$D_{KL}(q(\mathbf{s}_l)||(p(\mathbf{s}_l|\mathcal{D})) \qquad (6)$$

which is equivalent to maximizing,

$$\int q(\mathbf{s}_l)\log p(\mathcal{D}|\mathbf{s}_l)d\mathbf{s}_l - \mathrm{D}_{KL}[q(\mathbf{s}_l)||p(\mathbf{s}_l)], \qquad (7)$$

where $p(\mathcal{D}|\mathbf{s}_l)$ is the network's output probability given the values of the importance switch. We use eq. 7 as our optimization objective for optimizing $\boldsymbol{\phi}_l$ for each layer's importance switch.

Note that we can choose to perform the variational learning of each layer's importance switch sequentially from the input layer to the last layer before the output layer, or the learning of all importance switches jointly (the details on the difference between the two approaches can be found in the Sec. 4).

During the optimization, computing the gradient of eq. 7 with respect to $\boldsymbol{\phi}_l$ requires obtaining the gradients of the integral (the first term) and also the KL divergence term (the second term), as both depend on the value of $\boldsymbol{\phi}_l$. The KL divergence between two Dirichlet distributions can be written in closed form,

$$\begin{aligned}\mathrm{D}_{kl}[q(\mathbf{s}_l|\boldsymbol{\phi}_l)||p(\mathbf{s}_l|\boldsymbol{\alpha}_0)] &= \log\Gamma(\sum_{j=1}^{D_l}\boldsymbol{\phi}_{l,j}) - \\ &- \log\Gamma(D_l\alpha_0) - \sum_{j=1}^{D_l}\log\Gamma(\boldsymbol{\phi}_{l,j}) + D_l\log\Gamma(\alpha_0) \\ &+ \sum_{j=1}^{D_l}(\boldsymbol{\phi}_{l,j} - \alpha_0)\left[\psi(\boldsymbol{\phi}_j) - \psi(\sum_{j=1}^{D_l}\boldsymbol{\phi}_{l,j})\right],\end{aligned}$$

where $\boldsymbol{\phi}_{l,j}$ denotes the jth element of vector $\boldsymbol{\phi}_l$, Γ is the Gamma function and ψ is the digamma function.

Notice that the first term in eq. 7 requires broader analysis. As described in [7], the usual reparameterization trick, i.e., replacing a probability distribution with an equivalent parameterization of it by using a deterministic and differentiable transformation of some

[2] Notice that the Dirichlet parameters can take any positive value, $\alpha_i > 0$, however a sample from the Dirichlet distribution is a probability distribution whose values sum to 1

fixed base distribution[3], does not work. For instance, in an attempt to find a reparameterization, one could adopt the representation of a k-dimensional Dirichlet random variable, $\mathbf{s}_l \sim \mathrm{Dir}(\mathbf{s}_l|\boldsymbol{\phi}_l)$, as a weighted sum of Gamma random variables,

$$\mathbf{s}_{l,j} = y_j/(\sum_{j'=1}^{K} y_{j'}),$$

$$y_j \sim \mathrm{Gam}(\boldsymbol{\phi}_{l,j}, 1) = y_j^{(\boldsymbol{\phi}_{l,j}-1)} \exp(-y_j)/\Gamma(\boldsymbol{\phi}_{l,j}),$$

where the shape parameter of Gamma is $\boldsymbol{\phi}_{l,j}$ and the scale parameter is 1. However, this does not allow us to detach the randomness from the parameters as the parameter still appears in the Gamma distribution, hence one needs to sample from the posterior every time the variational parameters are updated, which is costly and time-consuming.

Implicit gradient computation. Existing methods suggest either explicitly or *implicitly* computing the gradients of the inverse CDF of the Gamma distribution during training to decrease the variance of the gradients (e.g., [21], [7], and [20]).

Analytic mean of Dirichlet random variable. Another computationally-cheap choice would be using the analytic mean of the Dirichlet random variable to make a point estimate of the integral $\int q_{\boldsymbol{\phi}_l}(\mathbf{s}_l) \log p(\mathcal{D}|\mathbf{s}_l) d\mathbf{s}_l \approx \log p(\mathcal{D}|\tilde{\mathbf{s}}_l)$, where $\tilde{\mathbf{s}}_{l,j} = \boldsymbol{\phi}_{l,j} / \sum_{j'=1}^{D_l} \boldsymbol{\phi}_{l,j'}$, which allows us to directly compute the gradient of the quantity without sampling from the posterior.

In our experiments, we examine the quality of posterior distributions learned with computing the gradients of the integral implicitly using the inverse CDF of the Gamma distribution, or with computing the gradients of the integral explicitly using the analytic mean of the Dirichlet random variable, in terms of the quality of learned architectures.

Note that as we add a probability vector (the importance switch) which sums to one, there is an effect of scaling down the activation values. However, once we learn the posterior distribution over the importance switch, we compress the network accordingly and then retrain the network with the remaining channels to recover to the original activation values. Our method is summarized in Algorithm 1. Also, note that step 3 of Algorithm 1 involves removing unimportant channels. Given the continuous values of posterior parameters, what is the cut-off that decides important channels from the rest at a given layer? In this paper, we search over sub-architectures at different pruning rates, where we select the important channels within those pruning rates as shown in Sec. 4. However, other ways, e.g., using the learned posterior uncertainty, can potentially be useful. We leave this as future work.

Algorithm 1 Dirichlet Pruning

```
Require: A pre-trained model, $\mathcal{M}_\theta$ (parameters are denoted by $\theta$).
Ensure: Compressed model $\hat{\mathcal{M}}_{\hat{\theta}}$ (reduced parameters are denoted by $\hat{\theta}$).
    Step 1. Add importance switches per layer to $\mathcal{M}_\theta$.

    Step 2. Learn the importance switches via optimizing eq. 7, with freezing $\theta$.
    Step 3. Remove unimportant channels according to the learned importance.
    Step 4. Re-train $\hat{\mathcal{M}}_{\hat{\theta}}$ with remaining channels.
```

4 EXPERIMENTS

In this section we apply the proposed method to create pruned architectures. The compression rates have been evaluated against a variety of existing common and state-of-the-art benchmarks, with the focus on probabilistic methods. We then also demonstrate how the important channels selected by our method may contain (human-perceivable) distinct visual features. The experiments are performed on three datasets, MNIST and FashionMNIST, which are used to train the LeNet-5 network, and CIFAR-10 used to train the ResNet-56, WideResNet-28-10 and VGG-16.

4.1 Variants of Dirichlet pruning

Dirichlet pruning is a flexible solution which allows for several variants. In the implementation of the importance switch parameter vector, the posterior distribution over switch via the variational inference objective as given in eq. 7 is evaluated. To compute the gradients of the integral (cross-entropy term) implicitly we use the samples from the inverse CDF of the Gamma distribution. For a given layer with n output channels we draw k samples of the importance switch vectors of length n. For Lenet-5 network we sample for $k = 50, 150, 300, 500$ and for VGG16 we sample for $k = 10, 20, 50, 100$ (the number of samples are provided in brackets when needed, e.g Dirichlet (300)).

In addition, we include the variant of the method where we compute the gradients of the integral explicitly using the analytic mean of the Dirichlet random variable (in the supplementary materials, we include an additional toy experiment which tests the difference between the two approaches). In the above approaches, we compute the importance switch vector

[3] For instance, a Normal distribution for z with parameters of mean μ and variance σ^2 can be written equivalently as $z = \mu + \sigma\epsilon$ using a fixed base distribution $\epsilon \sim \mathcal{N}(0, 1)$.

Method	Error	FLOPs	Params
Dirichlet (150)	1.1	168K	**6K**
Dirichlet (mean)	1.1	140K	**5.5K**
Dirichlet (joint)	1.1	158K	**5.5K**
BC-GNJ [27]	1.0	288K	15K
BC-GHS [27]	1.0	159K	9K
RDP [33]	1.0	117K	16K
FDOO (100K) [36]	1.1	**113K**	63K
FDOO (200K) [36]	1.0	157K	76K
GL [39]	1.0	211K	112K
GD [35]	1.1	273K	29K
SBP [32]	**0.9**	226K	99K

Table 1: The structured pruning of **LeNet-5**. The pruned network is measured in terms of the number of FLOPs and the number of parameters (Params). The proposed method outperforms the benchmark methods as far as the number of parameters is concerned and it produces the most optimal Params to FLOPs ratio.

for each layer separately. However, we are also able to train switch values for all the layers in one common training instance. This case is denoted by "joint" in brackets, e.g., Dirichlet (joint).

When computing the importance switch, we load the pretrained model in the first phase, and then add the importance switch as new parameters. We then fix all the other network parameters to the pretrained values and finetune the extended model to learn the importance switch. In compression process, we mask the subsets of features (both weights and biases, and the batch normalization parameters).

4.2 Compression

The goal of the neural network compression is to decrease the size of the network in such a way that the slimmer network which is a subset of the larger network retains the original performance but is smaller (which is counted in network parameters) and faster (counted in floating points operations or FLOPs). The bulk of the parameter load comes from the fully-connected layers and most of the computations are due to convolutional operations, and therefore one may consider different architectures for different goals.

We tackle the issue of compression by means of the Dirichlet pruning method in a way that the network learns the probability vector over the channels, that is where the support of the distribution is the number of channels. The channels that are given higher probability over the course of the training are considered more useful, and vice-versa. The probabilities over the channels can be ordered, and the channels which are given low probability can be pruned away. Subsequent to pruning, we retrain the network on the remaining channels.

Method	Error	FLOPs	Parameters
Dirichlet (ours)	**8.48**	**38.0M**	**0.84M**
Hrank [25]	8.77	73.7M	1.78M
BC-GNJ [27]	8.3	142M	1.0M
BC-GHS [27]	9.0	122M	0.8M
RDP [33]	8.7	172M	3.1M
GAL-0.05 [26]	7.97	189.5M	3.36M
SSS [17]	6.98	183.1M	3.93M
VP [43]	5.72	190M	3.92M

Table 2: **VGG-16** on CIFAR-10. Dirichlet pruning produces significantly smaller and faster models.

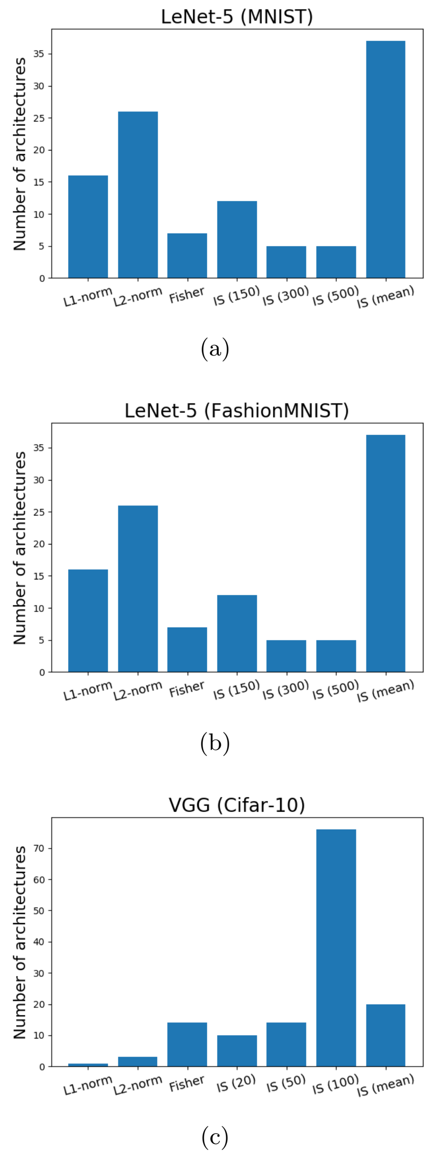
Method	Error	FLOPs	Parameters
Dirichlet (ours)	**9.13**	**13.64M**	**0.24M**
Hrank [25]	9.28	32.53M	0.27M
GAL-0.8 [26]	9.64	49.99M	0.29M
CP [14]	9.20	62M	-

Table 3: **ResNet-56** on CIFAR-10. Our method outperforms the recent methods, in particular when it comes to the model size (benchmark results come from the original sources). In the ResNet implementation, we use the approximation using the analytic mean.

Method	Error	Comp. Rate	Params
Dirichlet (ours)	4.5	52.2%	**17.4M**
L_0 ARM [24]	4.4	49.9%	18.3M
L_0 ARM [24]	**4.3**	49.6%	18.4M

Table 4: **WideResNet-28-10** on CIFAR-10. Compared to L_0-ARM, with a slight increase in the error rate, our method achieves the smallest number of parameters.

In the case of LeNet and VGG networks, we consider all the channels in every layer. In the case of residual networks each residual block consists of two convolutional layers. To preserve skip connection dimensionality in a similar fashion to [24], we prune the output channels of the first convolutional layer (equivalently input channels to the second layer). ResNet-56 consists of three sections with all convolutional layers having 16, 32 and 64 channels, respectively. Similarly, WideResNet-28-3 has 12 residual blocks (three sections of four blocks with 160, 320, 640 channels, respectively). We fix the number of channels pruned for each section. A finer approach could further bring better results.



(a)

(b)

(c)

Figure 3: Frequencies of best sub-architectures selected by each method. Considering 108 sub-architectures for LeNet-5 and 128 sub-architectures for VGG, the height of each bar describes the number of sub-architectures pruned by each method where a given method achieved the best test performance. We compare seven methods, including four variants of Dirichlet pruning, which we label by importance switch (IS). In all cases, our method dominantly performs over the largest set of sub-architectures, suggesting that the performance of our method is statistically significant.

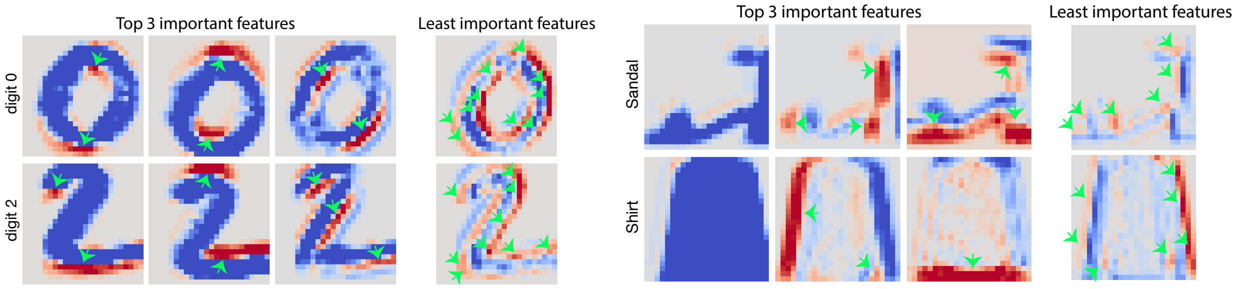
4.2.1 Compression rate comparison.

Table 1 presents the results of LeNet trained on MNIST, Table 2 the results of VGG trained on CIFAR-10. Moreover, we test two residual networks with skip connections, Table 3 includes the results of ResNet-56 and Table 4 demonstrates the results on WideResNet-28-10, both also trained on CIFAR-10. In the first test, we compare the results against the existing compression techniques, several of which are state-of-the-art Bayesian methods (we adopt the numbers from each of the papers). In the next subsection given the available codebase, we perform a more extensive search with magnitude pruning and derivative-based methods.

Note that our proposed ranking method produces very competitive compressed architectures, producing smaller (in terms of parameters) and faster (in terms of FLOPs) architectures with the similar error rates. In particular for LeNet, the compressed architecture has 5.5K parameters which is less than all the other methods, and 140K FLOPs which is third to RDP and FDOO(100K) that, however, have over three and ten times more parameters, respectively. The method works especially well on VGG producing an architecture which is smaller than others in the earlier layers but larger in later layers. This effectively reduces the number of required FLOPs compared to other state-of-the-art methods (44M in our case, two times less compared the second, HRank) for similar accuracy. The proposed methods are general and work for both convolutional and fully-connected layers, however they empirically show better results for convolutional layers. We believe that this behavior comes from the fact that these channels consist of a larger number of parameters and therefore are less affected by noise during SGD-based training (which gets averaged over these parameters), and therefore their importance can be measured more reliably.

4.2.2 Search over sub-architectures

In the second experiment for each method we verify method's pruning performance on a number of sub-architectures. We design a pool of sub-architectures with a compression rate ranging 20-60%. As mentioned earlier, some of the practical applications may require architectures with fewer convolutional layers to cut down the time and some may just need a network with smaller size. For Lenet-5 we use 108 different architectures and for VGG we test 128 architectures. We use the most popular benchmarks whose code is readily available and can produce ranking relatively fast. These are common magnitude benchmarks, L1- and L2-norms and the state-of-the art second derivative method based on Fisher pruning [4, 37]. Fig. 3 shows the number of times each method achieves superior results to the others after pruning it to a given sub-architecture. Dirichlet pruning works very well, in particular, for the VGG16 among over 80% of the 128 sub-architectures we considered, our method achieves better accuracy than others.



(a) **MNIST**

(b) **FashionMNIST**

Figure 4: Visualization of learned features for two examples from MNIST and FashionMNIST data for top three (the most important) features and bottom one (the least important) feature. Green arrows indicate where high activations incur. The top, most significant features exhibit strong activations in only a few *class-distinguishing* places in the pixel space. Also, these features exhibit the complementary nature, i.e., the activated areas in the pixel space do not overlap among the top 3 important features. On the other hand, the bottom, least significant features are more fainter and more scattered.

4.3 Interpretability

In the previous sections we describe the channels numerically. In this section, we attempt to characterize them in terms of visual cues which are more human interpretable. In CNNs, channels correspond to a set of convolutional filters which produce activations that can be visualized [41, 29]. Visualization of the first layer's feature maps provides some insight into how the proposed method makes its decisions on selecting important channels. As we presented the example from CIFAR-10 in Fig. 1, the feature maps of the important channels contain stronger signals and features that allow humans to identify the object in the image. In contrast, the less important channels contain features which can be less clear and visually interpretable.

In Fig. 4, we visualize feature maps produced by the first convolution layer of LeNet network given two example images from the MNIST and Fashion-MNIST, respectively. In contrast to the VGG network, almost all feature maps in LeNet allow to recognize the digit of the object. However, the important features tend to better capture distinguishing features, such as shapes and object-specific contour. In the MNIST digits, the learned filters identify *local parts* of the image (such as lower and upper parts of the digit '2' and opposite parts of the digit '0'). On the other hand, the most important feature in the FashionMNIST data is the overall shape of the object in each image, that is each class has different overall shape (e.g., shoes differ from T-shirts, bags differ from dresses).

The visualization of first layer's feature maps produced by the important channels helps us to understand why the compressed networks can still maintain a similar performance as the original immense networks. This seems to be because the compressed networks contain the core class-distinguishing features, which helps them to still perform a reliable classification even if the models are now significantly smaller. That being said, interpretability is a highly undiscovered topic in the compression literature. The provided examples illustrate the potential for interpretable results but a more rigorous approach is a future research direction.

5 Conclusion

Dirichlet pruning allows compressing any pre-trained model by extending it with a new, simple operation called *importance switch*. To prune the network, we learn and take advantage of the properties of Dirichlet distribution. Our choice for the Dirichlet distribution is deliberate. (a) A sample from Dirichlet distribution is a probability vector which sums to 1. (b) Careful choice of Dirichlet prior can encourage the sparsity of the network. (c) Efficient Bayesian optimization thanks to the closed-form expression of the KL-divergence between Dirichlet distributions. Thus, learning Dirichlet distribution allows to rank channels according to their relative importance, and prune out those with less significance. Due to its quick learning process and scalability, the method works particularly well with large networks, producing much slimmer and faster models. Knowing the important channels allows to ponder over what features the network deems useful. An interesting insight we gain through this work is that the features which are important for CNNs are often also the key features which humans use to distinguish objects.

Acknowledgments

The authors are supported by the Max Planck Society. Mijung Park is also supported by the Gibs Schüle Foundation and the Institutional Strategy of the University of Tübingen (ZUK63) and the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039B. Kamil Adamczewski is grateful for the support of the Max Planck ETH Center for Learning Systems.

Code

The most recent version of the code and the compressed models can be found at `https://github.com/kamadforge/dirichlet_pruning`. The stable version for reproducibility can also be found at `https://github.com/ParkLabML/Dirichlet_Pruning`.

REFERENCES

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[2] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. A closer look at structured pruning for neural network compression. *arXiv preprint arXiv:1810.04622*, 2018.

[5] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[6] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II, June 2003.

[7] Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 441–452. Curran Associates, Inc., 2018.

[8] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

[9] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354 – 377, 2018.

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[11] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[12] 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.

[13] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[14] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proc. ICCV*, pages 1389–1397, 2017.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[16] 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.

[17] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proc. ECCV*, pages 304–320, 2018.

[18] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[19] EK Ifantis and PD Siafarikas. Bounds for modified bessel functions. *Rendiconti del Circolo Matematico di Palermo Series 2*, 40(3):347–356, 1991.

[20] Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2235–2244, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[21] David A. Knowles. Stochastic gradient variational Bayes for gamma approximating distributions. *arXiv e-prints*, page arXiv:1509.01631, Sep 2015.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[24] Yang Li and Shihao Ji. l_0-arm: Network sparsification via stochastic binary optimization. *arXiv preprint arXiv:1904.04432*, 2019.

[25] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020.

[26] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proc. CVPR*, pages 2790–2799, 2019.

[27] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[28] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning Sparse Neural Networks through L_0 Regularization. *arXiv e-prints*, page arXiv:1712.01312, Dec 2017.

[29] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[31] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[32] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[33] Changyong Oh, Kamil Adamczewski, and Mijung Park. Radial and directional posteriors for bayesian neural networks. *arXiv preprint arXiv:1902.02603*, 2019.

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

[35] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[36] Raphael Tang, Ashutosh Adhikari, and Jimmy Lin. Flops as a direct optimization objective for learning sparse neural networks. *arXiv preprint arXiv:1811.03060*, 2018.

[37] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

[38] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[39] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[40] Kohei Yamamoto and Kurato Maeno. Pcas: Pruning channels with attention statistics for deep network compression. *arXiv preprint arXiv:1806.05382*, 2018.

[41] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[42] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[43] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proc. CVPR*, pages 2780–2789, 2019.