Channel selection using Gumbel Softmax

Charles Herrmann[1 [0000−0002−9576−9394]], Richard Strong Bowen[1 [0000−0002−9628−5471]], and Ramin Zabih[1,2 [0000−0001−8769−5666]]

[1] Cornell Tech
{cih, rsb, rdz}@cs.cornell.edu
[2] Google Research

Abstract. Important applications such as mobile computing require reducing the computational costs of neural network inference. Ideally, applications would specify their preferred tradeoff between accuracy and speed, and the network would optimize this end-to-end, using classification error to remove parts of the network. Increasing speed can be done either during training – e.g., pruning filters – or during inference – e.g., conditionally executing a subset of the layers. We propose a single end-to-end framework that can improve inference efficiency in both settings. We use a combination of batch activation loss and classification loss, and Gumbel reparameterization to learn network structure. We train end-to-end, and the same technique supports pruning as well as conditional computation. We obtain promising experimental results for ImageNet classification with ResNet (45-52% less computation).

Keywords: network sparsity, channel pruning, dynamic computation, Gumbel softmax

1 Pruning and conditional computation

Despite their great success [14, 24, 42], convolutional networks remain too computationally expensive for many important tasks. Modern architectures often struggle to run on standard desktop hardware, let alone mobile devices. These computational requirements pose a serious obstacle in settings constrained by latency, power, memory and/or compute; key examples include smartphones, robotics and autonomous driving. Considerable work has been put into exploring the tradeoffs between computation and performance. Popular approaches include expert-designed efficient networks like MobileNetV2 [39], and reinforcement learning to search for more efficient architectures [16, 54].

We focus on two longstanding lines of research: pruning [26, 37] and conditional computation [1, 49]. Pruning, in its earliest [26, 37] and modern [11, 21] forms, attempts to remove the least useful parts of the network. The goal is to leave a smaller network with comparable or better accuracy. A network with conditional computation runs lightweight tests that can choose to bypass larger blocks of computation that are not useful for the given input. Aside from benefits in inference-time efficiency [1], skipping computations can improve training

time or test performance [7, 20, 49], and can provide insight into network behavior [20, 49].

Our goal is to improve a neural network by trading off classification error and computation. End-to-end training is a key advantage of neural nets [25], but poses a technical challenge. Both pruning and conditional computation are categorical decisions which are not easy to optimize by gradient descent. However, Gumbel-Softmax (GS) [2, 10, 22, 33] gives a way to address this challenge.

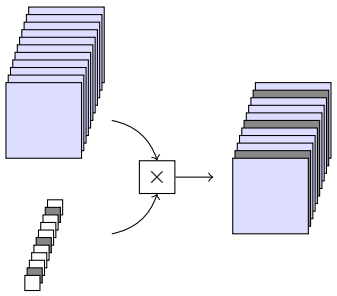


Fig. 1: The clean set of filter outputs (top left) are multiplied channel-wise by a vector of binary random variables (bottom left), which is learned during training. For conditional computation, the gating vector's entries depend upon the input at this layer, while for pruning they do not.

We focus on the ResNet [14] architecture, as it is the mainstay of current deep learning techniques for image classification. The general architecture of a prunable channel in a network is shown in Figure 1. The computation of a channel can potentially be skipped by sampling the gating vector of random variables. The associated probabilities are learned during training. Their distributions can be either depend on the layer's input, in which case we perform conditional computation, or be independent, in which case we perform pruning.

We propose a per-batch activation loss function, which allows the network to flexibly avoid computing certain filters and their resulting channels. This in turn supports useful tradeoffs between accuracy and inference speed. Per-batch activation loss, in combination with the Gumbel straight-through trick [22], encourages the gating vector's probabilities to *polarize*, that is, move towards 0 or 1. Polarization has proved to be beneficial [5, 44].

We summarize our contributions as follows:

- We explore conditional computation at the channel level and significantly outperform other techniques.
- We investigate the use of Gumbel soft-max for pruning a network at the channel-level in an end-to-end manner. We identify a mathematical property of the combination of our batch activation loss and Gumbel soft-max that encourages polarization.

– We demonstrate that a single technique can achieve significant results in both areas.

This paper is organized as follows. We begin by introducing notation and reviewing related work. Section 3 introduces and analyses our per-batch activation loss function and inference strategies, and discusses the role of polarization. Experimental results on ImageNet and CIFAR-10 are presented in Section 4, for both conditional computation and for pruning. Our best experimental results for pruning reduce computation by 51% on ResNet; our best results for conditional computation, reduce computation on ImageNet by 45–52% on ResNet. Additional experiments and more details are included in the supplemental material.

1.1 Gating neural networks

In order to learn a discrete structure such as a network architecture with the continuous method of stochastic gradient descent, we learn a probability distribution over structures, and minimize the expected loss. In this work, we learn whether or not to compute a channel. Let $\mathcal{G}$ be a set of gates indexed by i:

– Z_i, a 0-1 random variable which is 1 with probability p_i.
– g_i, a portion of the network which computes p_i.

When g and thus Z also depend on the input image j we write g_{ij}, p_{ij}, and Z_{ij}. Where g_i depends on the input we use the phrase "data-dependent"; where g_i does not, "data-independent". We use Gumbel Softmax and straight-through training [8, 22] to train g_i. To generate the vector of Z_is, we run each g_i and then sample. If $Z_i = 0$, the associated filter is not run, we simply replace the corresponding channel with a block of zeros. We use the straight-through trick: at training time during the forward pass, we use Z_i and during back-propagation, we treat Z_i as p_i. We define the "activation rate" of the batch as $\frac{1}{|\mathcal{G}||\mathcal{B}|} \sum_{0 \leq i \leq |\mathcal{G}|} \sum_{0 \leq j \leq |\mathcal{B}|} Z_{i,j}$ where $\mathcal{B}$ is the batch of inputs the network sees. This captures the fraction of the channels being computed for all gates over a batch. The "activation rate" of a gate i is $\frac{1}{|\mathcal{B}|} \sum_{0 \leq j \leq |\mathcal{B}|} Z_{i,j}$ This captures the fraction of time that the channel i is computed for the given batch.

1.2 Our loss

The intuition behind our loss is that we want to encourage the activation rate for each batch to approach a target rate hyperparameter t. Smaller values of t will correspond to less computation. Our batch activation loss is defined as

$$\mathcal{L}_B = \left(t - \frac{1}{|\mathcal{B}||\mathcal{G}|} \sum_{0 \leq i \leq |G|} \sum_{0 \leq j \leq |\mathcal{B}|} Z_{i,j} \right)^2 \tag{1}$$

2 Related work

Our technique allows us to learn a network with conditional computation (using data-dependent gates), or a smaller, pruned network (using data-independent gates). As such, we describe our relation to both fields, as well as related work on regularization.

2.1 Conditional computation

Cascaded classifiers [50] shorten computation by identifying easy negatives and have been adapted to deep learning [28, 51]. More recently, [19] and [34] both propose a cascading architecture which computes features at multiple scales and allows for dynamic evaluation, where at inference time the user can tradeoff speed for accuracy. Similarly, [48] adds intermediate classifiers and returns a label once the network reaches a specified confidence. [7, 9] both use the state of the network to adaptively decrease the number of computational steps during inference. [9] uses an intermediate state sequence and a halting unit to limit the number of blocks that can be executed in an RNN; [7] learns an image dependent stopping condition for each ResNet block that conditionally bypasses the rest of the layers in the block. [40] trains a large number of small networks and then uses gates to select a sparse combination for a given input. [3] selects the most-efficient network for a given input and also uses early-exit.

The most closely related work is AIG [49], which probabilistically gates individual layers during both training and inference, with a data-dependent gating computation. The major difference between AIG and our work is that they *specify* target rates for each gate, whereas we *learn* these values, by giving a target rate for the entire network. The reason for this difference is that AIG focuses on inducing specialization on the network, whereas we focus on improving the run-time of these networks. This focus on specialization is reflected in their loss, which has a target rate t_i for each gate i: $\mathcal{L}_{\text{G}} = \frac{1}{|\mathcal{G}|} \sum_{0 \leq i < |\mathcal{G}|} \left(t_i - \frac{1}{|\mathcal{B}|} \sum_{0 \leq j < |\mathcal{B}|} Z_{i,j} \right)^2$

As reported in their code[3], the target rates for each layer of ResNet-50 are [1, 1, 0.8, 1, t, t, t, 1, t, t, t, t, 1, 0.7, 1] for $t \in [0.4, 0.5, 0.6, 0.7]$. This loss function forces specialization since each gate learns to run at its target rate. However, constraining each gate identically is inflexible; for example, consider a dataset with two labels that are equally distributed. If the target rate t is different than 0.5, no layer will easily specialize to one of the labels. This value of t also determines the approximate speed of the final conditional network; networks trained with $t = 0.5$ will be about twice as fast as the baseline network. Yet AIG will push every layer with target rate t to specialize to run on half the data. This rules out many possible network configurations. Ideally, we want to pass in a single global target rate t for the network's speed and then allow the network to learn the optimal distribution of data for its gates. It can then choose to specialize individual gates on the the subsets which benefit the most from additional computation, and not be constrained to the gate's target rate.

[3] See https://github.com/andreasveit/convnet-aig

In addition, the loss that AIG uses cannot be adapted to network pruning, since it does not allow any the activation rate of any gate to approach 0 or 1 (a gate turning completely on or off). Additionally, modern network pruning is done on a channel-basis, which increases the number of gates from 17 for their layer version of ResNet-50 to thousands of gates for the channels of ResNet-18.

In summary, our loss enables the following improvements vis-a-vis AIG:

- Support for pruning. AIG only supports conditional computation.
- More granular control. AIG specifies per-gate target rates. Specifying per-gate target rates is infeasible at the scale of channels. Instead, our approach learns a rate for each gate, given a soft constraint on the full network.
- Improved performance. Our loss function gives the network more flexibility to configure the activation rates of individual gates. We find experimentally that our network can take advantage of this flexibility to make very different gate assignments (as demonstrated by comparing our Figure 4 and AIG's Figure 4). We also produce a much lower FLOPs count with comparable accuracy (as shown in Table 1).

We provide an experimental comparison with AIG in Section 4 and an ablation comparison in Section 4.4.

2.2 Pruning

Network pruning is another approach to decreasing computation time. Researchers initially attempted to determine the importance of specific weights [13,26] or hidden units [37] and remove those which are unimportant or redundant. Weight-based pruning on CNNs follows the same fundamental approach; [12] prunes weights with small magnitude and [11] incorporates these into a pipeline which also includes quantization and Huffman coding. Numerous techniques prune at the channel level, whether through heuristics [15, 18, 27] or approximations to importance [17, 36, 47]. [32] prunes using statistics from the following layer. [53] applies gates to a layer's weight tensors, sorts the weights during train time, and then sends the lowest to zero. Contemporary with our work, [52] uses a Taylor expansion rather than Gumbel to estimate the impact of opening or closing a gate; their technique prunes the network, but has no natural extension to conditional computation. Additionally, [29] suggests that the main benefits of pruning come primarily from the identified architecture.

Recently, several attempts have been made at doing channel-based pruning in an end-to-end manner. [21] adds sparsity regularization and then modifies stochastic Accelerated Proximal Gradient to prune the network end-to-end. Our work differs from [21] by using Gumbel Softmax to integrate the sparsity constraint into an additive loss which can be trained by any optimization technique; we use unmodified stochastic gradient descent with momentum (SGD), the standard technique for training classification.

Similarly, [31] uses the per-batch results of each layer to learn a per-layer "code". These codes are then used to learn a mask for the layer. As training

progresses, these masks are driven to be $0-1$ by increasing a sigmoid temperature term. The term in their loss function which trades off against computation time is similar to our per-batch activation loss defined in Equation 1. Their architecture does not use stochasticity or the Gumbel trick; we do not use a similar sigmoid temperature term, because we find that the variance term implicit in the loss is sufficient for pruning. See Section 3 for more details. We also provide an experimental comparison in Section 4.

2.3 Regularization and architecture search

Several regularization techniques, such as Dropout [46] and Stochastic Depth [20], have explored gating different parts of the network to make the final network more robust and less prone to over-fitting. Both techniques try to induce redundancy through probabilistically removing parts of the network during training. Dropout ignores individual units and Stochastic Depth skips entire layers. These techniques can be seen as gating units or layers, respectively, where the gate probabilities are hyperparameters.

In the Bayesian machine learning community, data-independent gating is used as both a form of regularization and for architecture search. Their regularization approaches can be seen as generalizing dropout by learning the dropout rates. [44] performs pruning by learning multipliers for weights, which are encouraged to be $0-1$ by a sparsity-inducing loss $w(1-w)$. [8] proposes per-weight regularization, using the straight-through Gumbel-Softmax trick. [43] uses a form of trainable dropout, learning a per-neuron gating probability. [45] learns sparsity at the weight level using a binary mask. They adopt a complexity loss (L_0 on weights) plus a sparsification loss similar to [44]. [30] extends the straight-through trick with a hard sigmoid to obtain less biased estimates of the gradient. They use a loss equal to the sum of Bernoulli weights, which is similar to a per-batch activation loss. [35] extends the variational dropout in [23] to allow dropout probabilities greater than a half.

Recently, several techniques have used binary gating or masking terms for architecture search. [41] uses Bernoulli random variables to dynamically learn network architecture elements, like connectivity, activation functions, and layers. Similarly, [4] learns a gating structure for convolutional blocks of different sizes, pools, etc. and proposes an end-to-end and reinforcement learning approach.

3 Technical considerations

A number of issues arise when applying our batch activation loss to speed inference. We begin with a discussion of polarization. We then describe training considerations followed by inference strategies. Finally we discuss our overall loss function and how to integrate gates into the ResNet architecture.

3.1 Gate polarization

Polarization plays a key role in several respects, and occurs extensively in our experimental results (see 4). In the framework laid out in Section 1.1, the p_i are a mechanism for learning discrete structures; in the independent case, a network architecture, and in the dependent case, an adaptive (or per-input) network architecture. The situation where the probabilities polarize corresponds to the continuous mechanism arriving at a discrete answer.

For data-independent gates, polarization corresponds to Z_i collapsing to always be either 0 or 1: in other words, each gate permanently chooses to run or skip its respective channel. In a perfectly polarized data-independent gating configuration some channels are never computed, and the network acts as a deterministic, pruned network. For data-dependent gates, polarization does not necessarily imply that the activation rate of a specific gate, $\frac{1}{|\mathcal{B}|}\sum_{0\leq j\leq|\mathcal{B}|} Z_{i,j}$ is 0 or 1; just that $\forall j$, $Z_{i,j}$ is either 0 or 1; under polarization, a gate's activation rate can have any value between 0 and 1. Conceptually, polarization means that for a given input, the decision whether or not to compute the channel is deterministic.

We observe that our batch activation loss has a property that actively encourages polarization in the independent case. Since $\mathcal{L}_B$ is a random variable, SGD and the straight-through trick can be seen as minimizing its expected value [22].

Property 1. In the independent case, the expected batch activation loss is 0 only if each g_i is polarized.

To see why this property holds, note that the expected activation loss is

$$\mathbb{E}[\mathcal{L}_B] = (t - \mathbb{E}[Q])^2 + \mathrm{Var}(Q) \quad \text{where} \quad Q = \frac{1}{|\mathcal{B}||\mathcal{G}|}\sum_{0\leq j<|\mathcal{B}|}\sum_{0\leq i<|\mathcal{G}|} Z_{i,j}$$

Clearly both terms in the expectation are non-negative and the second term (the variance) is only 0 at polarized values. The first term encourages the overall activation rate of the network to be close to t, but allows the activation rate of individual gates to vary. The second term generally encourages gate polarization.

3.2 Training considerations

As written in Equation 1, $\mathcal{L}_B$ is a random variable which we cannot backpropagate through. To solve this problem, we use the Gumbel reparameterization and straight-through training [8, 22] to train the network. We fixed the Gumbel softmax temperature at 1.0. We found that the straight-through trick ($Z_i \in \{0, 1\}$) typically had better performance than the soft version (e.g., Z_i being the Gumbel softmax of $(p, 1-p)$). For static pruning, g_j is simply two parameters that do not receive any input from the network and are directly passed to the Gumbel softmax. For dynamic pruning, g_j consists of an average pool across the image dimensions, a 1d convolution, a batch-norm, a ReLu, and then a final 1d convolution.

In image classification, the standard training regime includes global weight decay, which is equivalent to a squared L_2 norm on all weights in the network. We now describe an interaction between this regularization and gate polarization, which motivates a scaling of the weight decay parameter.

Generally, the Gumbel softmax trick reparameterizes the choice of a k-way categorical variable to a learning k (unnormalized) logits. In the specific $k = 2$ case for on-off gates, we learn two logits w_0 and w_1 for each gate. In the independent case, these two logits are themselves network parameters and therefore subject to weight decay. Given w_0 and w_1, the gate's on probability is just the sigmoid of their difference $p = \frac{1}{1+e^{w_0 - w_1}}$. The L_2 regularization implicitly adds the following to the loss: $w_0^2 + w_1^2 = \frac{1}{2}\left((w_0 + w_1)^2 + \ln\left(\frac{1-p}{p}\right)^2\right)$

The left hand term drives the logits towards $w_0 = -w_1$. We note that the logits $(w, -w)$ can produce any probability p. Since we are interested in the effect of weight decay on the learned gate probabilities, we focus primarily on the second term. It has the opposite of a polarizing effect: it reaches its minimum at $p = 0.5$. Since the weight decay loss is summed over all gates, this loss increases directly in proportion to the number of gates. We find that a weight decay parameter of 10^{-4} is suitable for a network of 10 to 20 gates. However, the implicit weight decay loss is a sum over probabilities whereas the variance term (Eq. 1) is an average. Therefore, we adopted a heuristic rule: for gating parameters, we divide the weight decay coefficient by the number of gates. Although the above analysis applies to the independent case, we found the same rule was effective for the dependent case.

3.3 Inference strategies

Once training has produced a deep network with stochastic gates, it is necessary to decide how to perform inference. The simplest approach is to leave the gates in the network and allow them to be stochastic during inference time. This is the technique that AIG uses. Experimentally, we observe a small variance so this may be sufficient for most use cases. One way to take advantage of the stochasticity is to create an ensemble composed of multiple runs with the same network. Then any kind of ensemble technique can be used to combine the different runs: voting, weighing, boosting, etc. In practice, we observe a bump in accuracy from this ensemble technique, though there is obviously a computational penalty.

However, stochasticity has the awkward consequence that multiple classification runs on the same image can return different results. There are several techniques to remove the stochasticity from the network. The gates can be removed, setting $Z = 1$ at test time. This is natural when viewing these gates as a regularization technique, and is the technique used by Stochastic Depth and Dropout. Alternately, inference can be made deterministic by using a threshold τ instead of sampling. Thresholding with value τ means that a layer will be executed if the learned probability p_i is greater than τ. This also allows the user some degree of dynamic control over the computational cost of inference. If the user passes in a very high τ, then fewer layers will activate and inference will be

faster. In our experiments, we set $\tau = \frac{1}{2}$. Note that we observe polarization for a large number of our per-batch experiments (particularly with data-independent gates). For a wide range of τ, thresholding leaves a network that behaves almost identically to a stochastic network; additionally, for a large number of τ the behavior of the thresholded network will be the same.

3.4 Architectural considerations

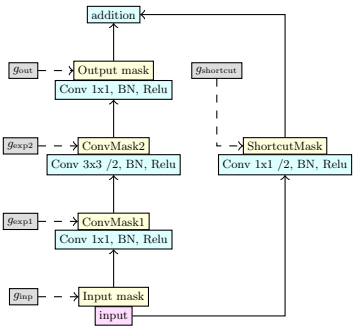


Fig. 2: Gating on ResNet Bottleneck Block

In Figure 2, we show the blocks for ResNet in its strided form. In Equation 1, each gate is given equal weight in the activation loss calculation. However for more complex gating schemes, not all gates control the same number of FLOPs (floating point operations per second). To compensate for this, we make a small change to batch activation loss; we change the activation loss to calculate the number of FLOPs using the $Z_{i,j}$. In this case, so $\mathcal{L}_{\mathcal{B}}$ takes the following form: $\mathcal{L}_{\text{FLOPs}} = \left(t - \frac{\#\text{ FLOPs}}{\text{Max }\#\text{ FLOPs}}\right)^2$. Our algorithm is to minimize the sum of this and and classification loss: $\mathcal{L} = \mathcal{L}_{\mathcal{C}} + \mathcal{L}_{\text{FLOPs}}$ where $L_{\mathcal{C}}$ is the classification loss.

4 Experiments

We implemented our method in PyTorch [38]. Our primary experiments centered around ResNet [14], running our resulting network on ImageNet [6]. Our main finding is that our techniques improve both accuracy and inference speed. We also perform an ablation study in order to better understand their performance.

4.1 Training parameters

For ResNet, we kept the same training schedule as AIG [49], and follow the standard ResNet training procedure: batch size of 256, momentum of 0.9, and weight decay of 10^{-4}. For the weight decay for gate parameters, we use $\frac{20}{|\mathcal{G}|} \cdot 10^{-4}$.

We train for 100 epochs from a pretrained model of the appropriate architecture with step-wise learning rate starting at 0.1, and decay by 0.1 after every 30 epochs. This is the same training schedule as [49]. We use standard training data-augmentation (random resize crop to 224, random horizontal flip) and standard validation (resize the images to 256×256 followed by a 224×224 center crop).

In practice, we noticed that many of our ResNet-50 models were not yet at convergence after this training schedule. In order to perform a fair comparison with [49], we did not train our data-dependent networks further. For our data-independent networks, we use the same training schedule as "fine-tune" in [15].

We observe that configurations with low activations rates for gates cause the batch norm estimates of mean and variance to be slightly unstable. Therefore before final evaluation for models trained with smaller batch size, we run training with a learning rate of zero and a large batch size for 200 batches in order to improve the stability and performance of the BatchNorm layers. Unless otherwise specified, we use deterministic inference with a threshold of 0.5.

4.2 Results on ImageNet

A graphical representation of the experimental results are in Figure 3a, as well as a detailed tabulated breakdown in Tables 2 and 1.

Pruning results Results are shown in Figure 3b and Table 2. We find that we can prune about 43% of the FLOPs with almost no loss of accuracy from the baseline model. In addition, we can achieve a higher Top-1 accuracy, 76.2, with 37% fewer FLOPs than ResNet-50. Compared to the natural competitor, AutoPruner[4] [31], with slightly fewer FLOPs, we have 0.8 higher accuracy. In additional, we perform nearly 0.7% better than the best baseline, Filter Pruning via Geometric Median [15], with a slightly smaller model (43% reduction compared to 42% reduction).

Conditional computation results Conditional computation results are shown in Figure 3a and Table 1. We find that we can skip 45-52% of the FLOPs from the baseline with comparable or even slightly better accuracy. ResNet-50 achieves a Top-1 accuracy of 76.13 with 4.028 FLOPs ($\times 10^9$). With target rate $t = .5$ we have a small improvement in accuracy (Top-1 accuracy of 76.3) at 2.21 FLOPs, which is 45% fewer. At $t = .4$ we have a small loss in accuracy (76.04) at 1.94 FLOPs, which is 52% fewer. The figures also show comparisons with AIG [49] (which is at the layer, rather than filter, granularity); we achieve a slightly higher accuracy with over 30% fewer FLOPs.

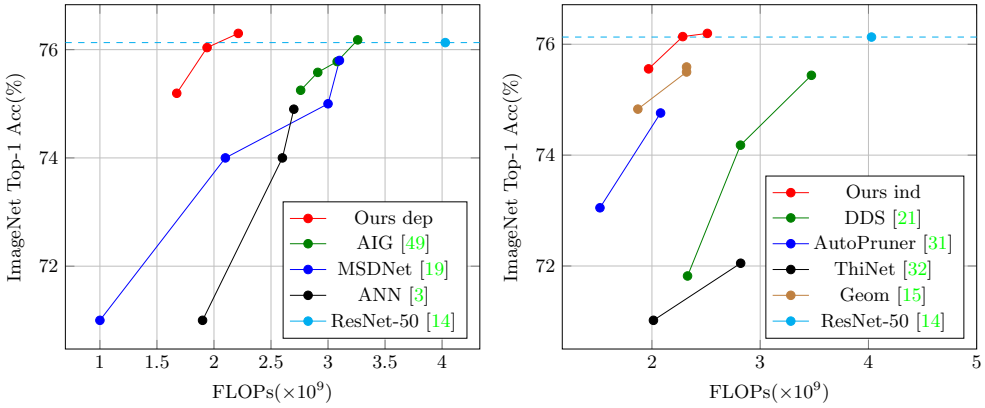
[4] Note that the number we use for their FLOPs is different from what they report. They report lower FLOPs for the baseline ResNet-50 architecture (3.8 GFLOPs versus our 4.028). To normalize the comparison, we added 0.2 GFLOPs to their results.

Model	Top-1 acc.(%)	Top-5 acc. (%)	FLOPs
ANN [3] at tradeoff 1	74.9	91.8	2.7
ANN [3] at tradeoff 2	74	91.8	2.6
MSDNet [19]-3.1	75.8	-	3.1
MSDNet [19]-3	75	-	3
MSDNet [19]-2.1	74	-	2.1
AIG [49] $t = 0.4$	75.25	92.39	2.76
AIG [49] $t = 0.5$	75.58	92.58	2.91
AIG [49] $t = 0.6$	75.78	92.79	3.08
AIG [49] $t = 0.7$	76.18	92.92	3.26
Ours dep $t = .5$	**76.30**	**93.01**	2.21
Ours dep $t = .4$	75.19	92.50	1.67
Ours dep $t = .3$	76.04	92.79	**1.94**

Table 1: Comparison of conditional computation on ImageNet-2012.

Depth	Model	Baseline top-1 acc.(%)	Accelerated top-1 acc. (%)	Top-1 acc ↓	Top-5 acc ↓	FLOPs ↓ (%)
18	Geom [15] (only 30%)	70.28	68.34	1.94	1.10	41.8
	Geom [15] (mix 30%)	70.28	68.41	1.87	1.15	41.8
	Ours ind	70.28	**68.88**	**1.40**	**0.97**	**43.9**
34	Geom [15] (only 30%)	73.92	72.54	1.38	**0.49**	41.1
	Geom [15] (mix 30%)	73.92	72.63	1.29	0.54	41.1
	Ours ind	73.92	**72.78**	**1.14**	0.69	**51.1**
50	DDS-41	76.13	75.44	0.69	2.25	13.77
	DDS-32	76.13	74.18	1.95	1.04	30.0
	DDS-26	76.13	71.82	4.31	0.29	42.17
	ThiNet-70	72.88	72.04	0.84	3.08	36.7
	AutoPruner 0.5	76.13	74.76	1.37	0.79	48.36
	Geom (only 30%)	76.15	75.59	0.56	0.24	42.2
	Geom (mix 30%)	76.15	75.50	0.65	0.21	42.2
	Geom (only 40%)	76.15	74.83	1.32	0.55	**53.5**
	Ours ind $t = .5$	76.13	**76.20**	**-0.07**	**-0.2**	37.68
	Ours ind $t = .4$	76.13	76.14	-0.01	-0.04	43.39
	Ours ind $t = .3$	76.13	75.56	0.56	0.36	51.3

Table 2: Comparison of pruned ResNet on ImageNet-2012. Acc ↓ is the decrease in accuracy between the accelerated model and the baseline mode; lower is better. Baseline numbers are listed because different researchers' implementations vary; note that our compressed model for ResNet-34 is smaller than that of Geom [15].



(a) Conditional computation results for ResNet-50

(b) Pruning results for ResNet-50

Fig. 3: Selected experimental results for ImageNet.

Variant	FLOP %	Baseline top-1 acc (%)	Accelerated top-1 acc (%)	Top-1 Δ
Conditional computation on ResNet110				
AIG-110 $t = .8$	82%	93.39	94.24	1% ↑
Ours dep $t = .6$	65%	93.39	94.36	1% ↑
Pruning on ResNet-56				
AMC	50%	92.8	91.9	1.0% ↓
Ours ind $t = .5$	50%	93.86	93.31	0.4% ↓

Table 3: CIFAR-10 results. The FLOPs is reported as a percentage of the original model and accuracy is reported for the baseline and accelerated models. Note that our ResNet-56 baseline is more accurate than AMC's ResNet-56 baseline.

Model	Baseline FLOPs (10^9)	Accelerated FLOPs (10^9)	FLOPs Δ	Top-1 acc. (%)
AIG-50 $t = 0.6$	4.028	3.08	76.5%	75.78
Ours data-dep. layer $t = 0.5$	4.028	2.72	67.5%	75.78
Ours data-dep. filter $t = 0.4$	4.028	1.94	48.1%	76.07

Table 4: Layer vs Filter granularity for gating. FLOPs Δ is calculated from baseline ResNet50 architecture (lower is better).

4.3 Results on CIFAR-10

We report results on CIFAR-10 on several architectures and compare to other techniques; see Figure 3. Using conditional computation, we obtain higher accuracy on ResNet-110, 94.36, with 65% fewer FLOPs. Compared to AIG, we obtain higher accuracy with 20.7% fewer FLOPs. Using pruning on ResNet-56, we can reduce the number of FLOPs by 50% with only a small decrease in final accuracy, 93.31. Compared with AMC, we have a smaller decrease in accuracy at the same FLOPs reduction. Additional results are included in the supplemental.

4.4 Analysis and ablation studies

Filter vs layers Our proposed techniques can be used on a layer basis; our per-batch activation loss, in combination with the Gumbel, still provides strong performance. In general, operating at filter granularity rather than layers provides a substantial boost: roughly 20% improvement in FLOPs at the same accuracy. Results are shown in Table 4. For pruning (data-independent gates), moving from layer to filter granularity results in a 28% improvement in FLOPs for a similar accuracy. For conditional computation (data-dependent gates), we can do an even more detailed ablation study since the primary difference between AIG [49] and our result are the batch activation loss and the filter granularity. Overall, batch activation loss provides approximately a 12% boost over AIG and filter granularity provides an additional 27% improvement over the layer-based version of our technique.

Specialization results In Figure 4, we show the gate activations for our layer-based data-dependent model. We observe higher levels of specialization than AIG. While AIG's model specializes primarily on manmade vs non-manmade objects, we specialize on more granular category types. The first layer runs only on cats and dogs, while the second layer runs primarily on fish and lizards. Note, that the specialization shown in AIG is a direct result of the chosen target rate. Their target rate of $t = 0.5$ causes each layer in their model to specialize on one half of the dataset. Since our target rate is for the entire network, each layer in our model is able to specialize on whatever subset it wants. In fact, our specialized layers (cats and dogs; lizards and fish) suggest that working on smaller, more specific subsets can help the model's accuracy.

5 DenseNet extensions

There are a number of natural extensions to our work that we have explored. Here, we focus on the use of probabilistic gates to provide an early exit, when the network is certain of the answer. We are motivated by MSDNet [19], which investigated any-time classification. We explore early exit on both ResNet and DenseNet; however, consistent with [19], we found that ResNet tends to degrade with intermediate classifiers while DenseNet does not. Following [48] we place

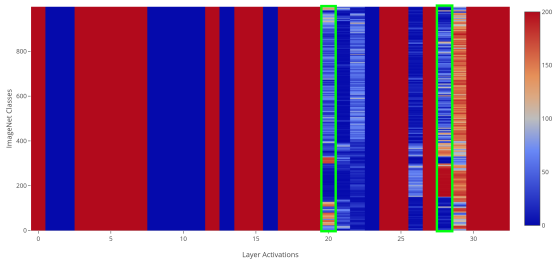


Fig. 4: Gate activations for a layer-based conditional computation model with our batch-activation loss. Two of the layers are highlighted in green. These layers depict a higher level of specialization than those shown in AIG; the first layer runs on cats and dogs, while the second layer runs primarily on fish and lizards.

gates and intermediate classifiers at the end of each dense block. At each gate, the network makes a decision as to whether the instance can be successfully classified. Results are shown in Table 5. These early exit gates make good decisions regarding which instances to classify early. More details are in the supplementals.

	Images selected (%)	Top-1 acc. (%) (all images)	Top-1 acc. (%) (selected images)
Block 1	28.71%	81.36	96.37
Block 2	11.56%	93.35	98.53
Final Block	59.74%	94.19	92.63

Table 5: DenseNet on CIFAR-10 with early exit. For early classifiers, the accuracy on the images that the network selects is higher than the accuracy on all images. This suggests that the gates are learning to recognize "easy" examples.

6 Conclusion

We show an end-to-end trainable system for selecting channels using Gumbel soft-max. We propose a single framework that can handle both pruning and conditional computation at the channel level. Our novel batch loss, combined with the Gumbel trick for making categorical gate decisions, shows strong quantitative speedups over the baseline, improving the FLOPS-accuracy Pareto frontier.

Acknowledgments. We thank Andreas Veit and Serge Belongie for their invaluable insights on AIG and several reviewers for helpful comments. This work was generously supported by Google Cloud, without whose help it could not have been completed. It was funded by NSF grant IIS-1447473, by a gift from Sensetime, and by a Google Faculty Research Award.

References

1. Bengio, Y.: Deep learning of representations: Looking forward. CoRR **abs/1305.0445** (2013), http://arxiv.org/abs/1305.0445 1
2. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR **abs/1308.3432** (2013), http://arxiv.org/abs/1308.3432 2
3. Bolukbasi, T., Wang, J., Dekel, O., Saligrama, V.: Adaptive neural networks for efficient inference. In: ICML. pp. 527–536 (2017) 4, 11, 12
4. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. arXiv:1812.00332 (2018) 6
5. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: NIPS. pp. 3123–3131 (2015) 2
6. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. pp. 248–255. Ieee (2009) 9
7. Figurnov, M., Collins, M.D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D.P., Salakhutdinov, R.: Spatially adaptive computation time for residual networks. In: CVPR (2017) 2, 4
8. Gal, Y., Hron, J., Kendall, A.: Concrete dropout. In: Advances in Neural Information Processing Systems. pp. 3581–3590 (2017) 3, 6, 7
9. Graves, A.: Adaptive computation time for recurrent neural networks. CoRR **abs/1603.08983** (2016) 4
10. Gumbel, E.J.: Statistical theory of extreme values and some practical applications. NBS Applied Mathematics Series **33** (1954) 2
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015) 1, 5
12. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015) 5
13. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. In: Advances in neural information processing systems. pp. 164–171 (1993) 5
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016) 1, 2, 9, 12
15. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4340–4349 (2019) 5, 10, 11, 12
16. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: Automl for model compression and acceleration on mobile devices. In: ECCV. pp. 784–800 (2018) 1
17. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: ICCV (2017) 5
18. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016) 5
19. Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., Weinberger, K.Q.: Multi-scale dense convolutional networks for efficient prediction. CoRR, abs/1703.09844 **2** (2017) 4, 11, 12, 13

20. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: ECCV. pp. 646–661. Springer (2016) 2, 6
21. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. ECCV (2018) 1, 5, 12
22. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. In: ICLR (2017), arXiv preprint arXiv:1611.01144 2, 3, 7
23. Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) NIPS, pp. 2575–2583. Curran Associates, Inc. (2015), http://papers.nips.cc/paper/5666-variational-dropout-and-the-local-reparameterization-trick.pdf 6
24. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012), http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf 1
25. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (May 2015), https://doi.org/10.1038/nature14539 2
26. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990) 1, 5
27. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. ICLR (2017) 5
28. Li, H., Lin, Z., Shen, X., Brandt, J., Hua, G.: A convolutional neural network cascade for face detection. In: CVPR. pp. 5325–5334 (2015) 4
29. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: ICLR (2019), https://openreview.net/forum?id=rJlnB3C5Ym 5
30. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. ICLR (2017), arXiv preprint arXiv:1712.01312 6
31. Luo, J.H., Wu, J.: Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. arXiv preprint arXiv:1805.08941 (2018) 5, 10, 12
32. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. ICCV (2017), arXiv preprint arXiv:1707.06342 5, 12
33. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712 (2016) 2
34. McGill, M., Perona, P.: Deciding how to decide: Dynamic routing in artificial neural networks. In: ICML (2017), arXiv preprint arXiv:1703.06217 4
35. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: Precup, D., Teh, Y.W. (eds.) ICML. vol. 70, pp. 2498–2507. PMLR (06–11 Aug 2017), http://proceedings.mlr.press/v70/molchanov17a.html 6
36. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. ICLR (2016), arXiv preprint arXiv:1611.06440 5
37. Mozer, M.C., Smolensky, P.: Skeletonization: A technique for trimming the fat from a network via relevance assessment. In: Advances in neural information processing systems. pp. 107–115 (1989) 1, 5
38. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS-W (2017) 9
39. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR. pp. 4510–4520 (2018) 1

40. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., Dean, J.: Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538 (2017) 4
41. Shirakawa, S., Iwata, Y., Akimoto, Y.: Dynamic optimization of neural network structures using probabilistic modeling. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018) 6
42. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014) 1
43. Srinivas, S., Babu, R.V.: Generalized dropout. arXiv preprint arXiv:1611.06791 (2016) 6
44. Srinivas, S., Babu, V.: Learning neural network architectures using backpropagation. In: BMVC. pp. 104.1–104.11 (September 2016). https://doi.org/10.5244/C.30.104, https://dx.doi.org/10.5244/C.30.104 2, 6
45. Srinivas, S., Subramanya, A., Venkatesh Babu, R.: Training sparse neural networks. In: CVPR Workshops (July 2017) 6
46. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(1), 1929–1958 (2014) 6
47. Suau, X., Zappella, L., Palakkode, V., Apostoloff, N.: Principal filter analysis for guided network compression. arXiv preprint arXiv:1807.10585 (2018) 5
48. Teerapittayanon, S., McDanel, B., Kung, H.: Branchynet: Fast inference via early exiting from deep neural networks. In: ICPR. pp. 2464–2469. IEEE (2016) 4, 13
49. Veit, A., Belongie, S.: Convolutional networks with adaptive inference graphs. In: ECCV (2017), https://github.com/andreasveit/convnet-aig 1, 2, 4, 9, 10, 11, 12, 13
50. Viola, P., Jones, M.J.: Robust real-time face detection. International Journal of Computer Vision **57**(2), 137–154 (2004) 4
51. Yang, F., Choi, W., Lin, Y.: Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In: CVPR. pp. 2129–2137 (2016) 4
52. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 2130–2141 (2019) 5
53. Zhu, M., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878 (2017) 5
54. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR. pp. 8697–8710 (2018) 1