An end-to-end approach for speeding up neural network inference

Charles Herrmann
Cornell University
cih5@cornell.edu

Richard Strong Bowen
Cornell University
rsb349@cornell.edu

Ramin Zabih
Cornell University & Google
rdz@cs.cornell.edu

April 25, 2019

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 [27, 38, 45]. Increasing speed can be done either during training – e.g., pruning filters [28] – or during inference – e.g., conditionally executing a subset of the layers [50]. We propose a single end-to-end framework that can improve inference efficiency in both settings. We introduce a batch activation loss and use Gumbel reparameterization to learn network structure [23, 50]. We train end-to-end against batch activation loss combined with classification loss, and the same technique supports pruning as well as conditional computation. We obtain promising experimental results for ImageNet classification with ResNet [14] (45-52% less computation) and MobileNetV2 [40] (19-37% less computation).

1 Pruning and conditional computation

Despite their great success [14, 25, 43], 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 [40], and reinforcement learning to search for more efficient architectures [15, 54].

We focus on two longstanding lines of research: pruning [27, 38] and conditional computation [1, 50]. Pruning, in both its earliest [27, 38] and modern [11, 22] incarnations, attempts to remove the least beneficial parts of the network. The goal is to leave a smaller network with comparable or even 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, 21, 50], and can provide insight into network behavior [21, 50].

Our goal is to improve a neural network by trading off classification error and computation. End-to-end training is a key advantage of modern neural networks [26], but poses a significant technical challenge. Both pruning and conditional computation are categorical decisions which are not easy to optimize via gradient descent. However, Gumbel-Softmax (GS) [2, 10, 23, 34] provides a way to address this challenge.

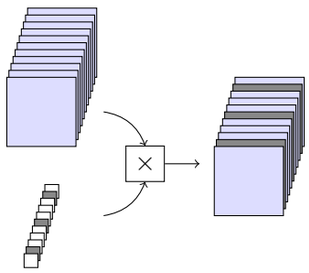


Figure 1: The clean set of filter outputs (top left) are multiplied channel-wise by a vector of binary random variables (bottom left), which is learned during training. For conditional computation, the gating vector's entries depend upon the input at this layer, while for pruning they do not.

We focus on the ResNet [14] and MobileNetV2 [40] architectures since these are 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 [23], encourages the gating vector's probabilities to *polarize*, that is, move towards 0 or 1. Polarization has proved to be beneficial [5, 45].

This paper is organized as follows. We begin by introducing notation and briefly 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 are for conditional computation, where we reduce computation on ImageNet by 45–52% on ResNet and 19–37% on MobileNetV2. Additional experiments and more details are included in the supplemental material appendix.

1.1 Gating neural networks

In order to learn a discrete structure such as a network architecture with the fundamentally continuous method of stochastic gradient descent, we learn a probability distribution over structures, and minimize the expected loss. Following [50], 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 [8,23] 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 and 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 refer to $\frac{1}{|\mathcal{G}||\mathcal{B}|}\sum_{0\le i\le|\mathcal{G}|}\sum_{0\le j\le|\mathcal{B}|}Z_{i,j}$ as the "activation rate" of the batch; this captures the fraction of the channels being computed for all gates over a batch. The "activation rate" of a gate i is $\frac{1}{|\mathcal{B}|}\sum_{0\le j\le|\mathcal{B}|}Z_{i,j}$; this captures the fraction of time that the channel i is computed for the given batch.

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 [51] shortened computation by identifying easy negatives and have been adapted to deep learning [29, 52]. More recently, [18] and [35] 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, [49] 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. [41] 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.

Our work was motivated by AIG [50], which probabilistically gates individual layers during both training and inference, with a data-dependent gating computation. One important difference is that their loss function encourages each layer to specialize whereas ours measures overall speed. We defer further discussion until Section 3.1. We provide an experimental comparison with AIG in Section 4 and an ablation comparison in Section 4.5.1.

2.2 Pruning

Network pruning is another approach to decreasing the computation time. Researchers, initially, attempted to determine the importance of specific weights [13, 27] or hidden units [38] 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 [17, 28] or approximations to importance [16, 37, 48]. [33] prunes using statistics from the following layer. [53] applies gates a layer's weight tensors, sorts the weights during train time, and then sends the lowest to zero. Additionally, [30] suggests that the main benefits of pruning come primarily from the identified architecture.

Recently, several attempts have been made to do channel-based pruning in an end-to-end manner. [22] adds sparsity regularization and then modifies stochastic Accelerated Proximal Gradient to prune the network in an end-to-end fashion. Our work differs from [22] 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, [32] 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 [47] and Stochastic Depth [21], 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. [45] 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. [44] uses a form of trainable dropout, learning a per-neuron gating probability. [46] learns sparsity at the weight level using a binary mask. They adopt a complexity loss which is L_0 on weights, plus a sparsification loss similar to [45].

[31] 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. [36] extends the variational dropout in [24] to allow dropout probabilities greater than a half.

Recently, several techniques have used binary gating or masking terms for architecture search. [42] 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 approach

In this section, we discuss our proposed per-batch activation loss function, and discuss some of its technical consequences. 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

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

A number of issues arise when applying our batch activation loss to speed inference. We begin with a discussion of polarization, which also provides a useful contrast with AIG [50]. We then describe training considerations followed by inference strategies. Finally we discuss our overall loss function and how to integrate gates into the ResNet and MobileNet architectures.

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.

We observe that our batch activation loss has a property that actively encourages polarization. Since $\mathcal{L}_B$ is a random variable, SGD and the straight-through trick can be seen as minimizing its expected value [23].

Property 3.1 *The expected batch activation loss is* 0 *only if each* g_i *is polarized.*

The expected activation loss is

$$Q = \frac{1}{|\mathcal{B}||\mathcal{G}|} \sum_{0 \le j < |\mathcal{B}|} \sum_{0 \le i < |\mathcal{G}|} Z_{i,j}$$

$$\mathbb{E}[\mathcal{L}_B] = (t - E[Q])^2 + \text{Var}(Q)$$

Clearly both terms 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.

AIG uses a target activation rate for each gate. Given a target rate $t \in [0, 1]$ this is

$$\mathcal{L}_{\text{G}} = \frac{1}{|\mathcal{G}|} \sum_{0 \le i < |\mathcal{G}|} \left(t - \frac{1}{|\mathcal{B}|} \sum_{0 \le j < |\mathcal{B}|} Z_{i,j} \right)^2$$

Their loss function encourages each layer to be run at the target rate t, which is much less flexible than per-batch activation loss. For example, imagine a $2k$ gate network with $t = .5$. Under AIG's per-gate loss, the only network with expected loss of 0 is one where each gate opens exactly for half of any given batch. However, using our polarization-encouraging batch activation loss, there are at least $\binom{2k}{k}$ networks with expected loss of 0. In practice, AIG used a hand-set target rate for each layer[1]; however, this is impractical when $|\mathcal{G}|$ is in the hundreds or thousands.

3.2 Training considerations

As written in Equation 1, $\mathcal{L}_B$ is a random variable which we cannot back-propagate through. To solve this problem, we use the Gumbel reparameterization and straight-through training [8, 23] 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 $(p, 1 - p)$).

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:

$$w_0^2 + w_1^2 = \frac{1}{2}\left((w_0 + w_1)^2 + \ln\left(\frac{1-p}{p}\right)^2\right) \tag{2}$$

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 (Eq. 2) 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 small 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 $\tau = \frac{1}{2}$.

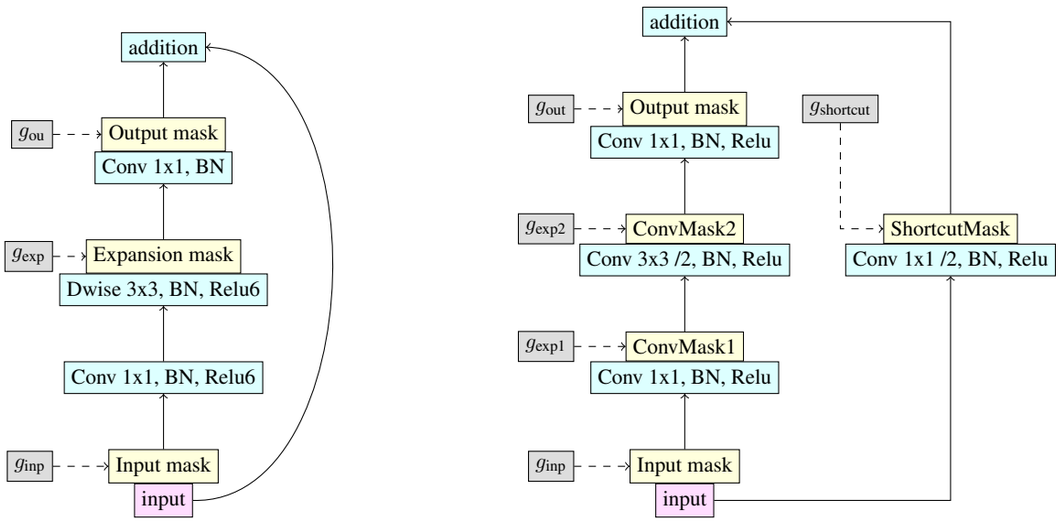
Note that we observe polarization for a large number of our per-batch experiments (particularly with data-independent gates). In this situation, for a wide range of τ, thresholding leaves a network that behaves almost identically to the stochastic network; additionally, for a large number of τ the behavior of the thresholded network will be the same.

3.4 Architectural considerations

In Figure 2, we show the blocks for MobileNetV2 and ResNet in their strided form. ResNet experiments are based on ResNet-50 and MobileNetV2 experiments are based on MobileNetV2 with a width parameter of 1.0.

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 pxoint 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 $Z_{i,j}$. For example, consider a single MobileNetV2 Inverted Residual block operating on a tensor of height H and width W. Let s be the block's stride. The input, expansion, and output channel masks as shown in Figure 2 have respectively C_{in}, C_{exp}, and C_{out} active channels. Then executing this block requires the following number of FLOPs:

[1] For example, for ResNet-50, AIG used the following target rates for the layers [1, 1, 0.8, 1, t, t, t, 1, t, t, t, t, t, 1, 0.7, 1] [50]



(a) Gating on MobileNetV2 Block

(b) Gating on ResNet Bottleneck Block

Figure 2: Architectures for channel gating.

$$HWC_{\text{exp}}(C_{\text{in}} + \frac{9}{s^2} + \frac{C_{\text{out}}}{s^2})$$

Note that this directly mirrors the FLOPs calculation for a MobileNetV2 block. Also note that each C_{exp} is the sum over $Z_{i,j}$ that mask the exp channels. We then sum over all blocks and take the mean over all instances in a batch. So $\mathcal{L}_{\mathcal{B}}$ takes the following form:

$$\mathcal{L}_{\text{FLOPs}} = \left(t - \frac{\text{\# FLOPs}}{\text{Max \# FLOPs}}\right)^2$$

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 [39]. Our primary experiments centered around ResNet [14] and MobileNetV2 [40], 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 [50], and followed 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 after every 30 epochs decay by 0.1. Note that this is the same training schedule as [50].

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 [50], we did not train our data-dependent networks further. For our data-independent networks, we run for an additional 30 epochs; note that we experimentally compare against methods trained to convergence.

For MobileNetV2, we use a similar training schedule to those reported in PyTorch repositories: SGD with batch size of 256, momentum of 0.9, weight decay of 10^{-4}. For the weight decay for gate parameters, we us $\frac{20}{|\mathcal{G}|} \cdot 10^{-4}$. We use a starting learning rate of 0.5 and decay the learning rate by 0.99 until convergence, approximately 700 epochs. We tested a decay rate of 0.98 and 350 epochs, and obtained the same accuracy with a small number of additional flops. Note that we train separately for 96-by-96 inputs and 224-by-224 inputs, producing separate architectures for the separate input sizes.

We use standard training data-augmentation (random resize crop to 224 and random horizontal flip) and standard validation (rescale the images to 256×256 followed by a 224×224 center crop). For input resizing, we follow the TensorFlow "fast_mode" setting[2] which uses bilinear resizing at both training and test time.

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 batchsize, 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.

[2] https://github.com/tensorflow/models/blob/master/research/slim/preprocessing/inception_preprocessing.py

4.2 Granularity

To decrease training time, we typically group sets of filters together under a single gate. We use the term "granularity" to refer to the number of filters that a single gate controls; high granularity indicates that each gate controls a small number of filters. Naturally, the highest possible granularity is a single gate for each filter. Experimentally, we observe that lower granularities have better earlier performance and seem to result in networks with higher FLOPs and slightly higher accuracy. Due to the lower training time required, the experimental results in Section 4.3 use the following low granularity settings:

t	c	n	s	g_{inp} granularity	g_{out} granularity	g_{exp} granularity
1	16	1	1	8	8	16
6	24	2	2	8	8	24
6	32	3	2	8	16	24
6	64	4	2	8	16	32
6	96	3	1	16	32	48
6	160	3	2	16	32	64
6	320	1	1	32	64	64

For ResNet-50, each gate controls 16, 32, 64 or 128 filters in the four layers' expansions, and half that in the contractions.

Note that the gating scheme roughly tries to keep the number of FLOPs controlled by a single gate constant; as such, the granularity in earlier layers tends to be higher than the granularity in later layers. Additionally, for non-residual layers, we also force a small constant number of gates to always return 1 (always run the computation). For the low granularities reported, we keep 1 gate open for non-residual layers.

4.3 Results on ImageNet

Tabulated experimental results for our techniques and the main competitors are in figure 4. Selected results are graphed in figure 3. In general we got the best results from conditional computation, but the pruning experimental results are also promising.

4.3.1 Conditional computation results

ResNet-50 Conditional computation results are shown in Figure 3a and 4b. 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 Prec@1 of 76.13 with 4.028 FLOPs ($\times 10^9$). With target rate $t = .5$ we have a small improvement in accuracy (Prec@1 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 [50] (which is at the layer, rather than channel, granularity). Compared to AIG, we achieve a slightly higher accuracy with over 30% fewer FLOPs.

MobileNetV2 Results are shown in Figures 3c and 3d, where we achieve a speedup of 19-37% . MobileNet's reported results are somewhat sparse over a wide range of FLOPs, so to compute our relative speed we interpolated between consecutive points as shown in Figure 3c.

For input size of 96, we obtain a Prec@1 accuracy of 59.5 at 28.3 FLOPs ($\times 10^6$). Interpolating between the MobileNetV2 points with accuracy just above and below ours gives an estimated MobileNetV2 speed of 47.4 FLOPs at our accuracy. We thus reduced the computation by 37% at the same accuracy, as shown by the dashed arrow in Figure 3c. At input size 224 our results are also promising. Applying the same interpolation scheme, our speedup ranges from 19-29%.

4.3.2 Pruning results

ResNet-50 Results are shown in Figures 3b and 4a. We find that we can prune about 35% of the FLOPs with almost no loss of accuracy from the baseline model. We achieve a higher Top-1 accuracy, 76.2, with 37% fewer FLOPs than ResNet-50, using 2.51 FLOPs (10^9). Compared to the natural competitor, AutoPruner[3] [32], with slightly fewer FLOPs, we have 0.8 higher accuracy.

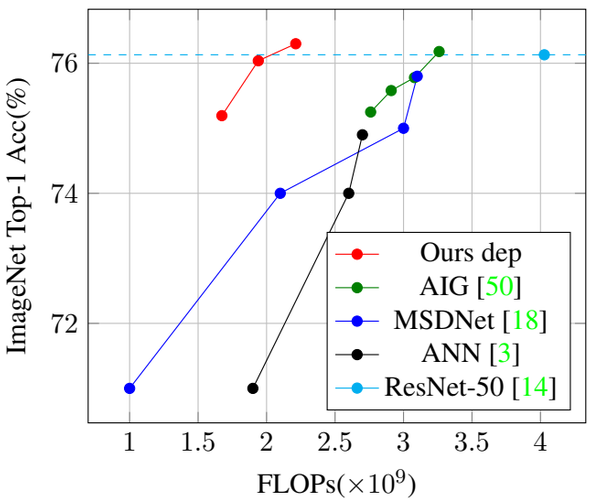
MobileNetV2 Pruning results are shown in Figures 3c and 3d. For input size 224, we achieve a slight improvement over the baseline; and for 96, we reduce computation by approximately 30%.

4.4 Results on CIFAR-10

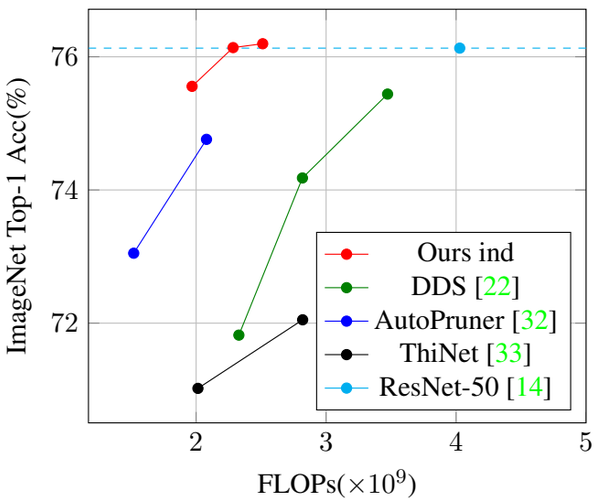
We report results on CIFAR-10 on several architectures and compare to other techniques; see Figure 5. Using conditional computation, we obtain higher accuracy on ResNet-110, 94.36, with 65% fewer FLOPs. Compared to AIG, we obtain higher accuracy with 20.7% fewer FLOPs.

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 material appendix.

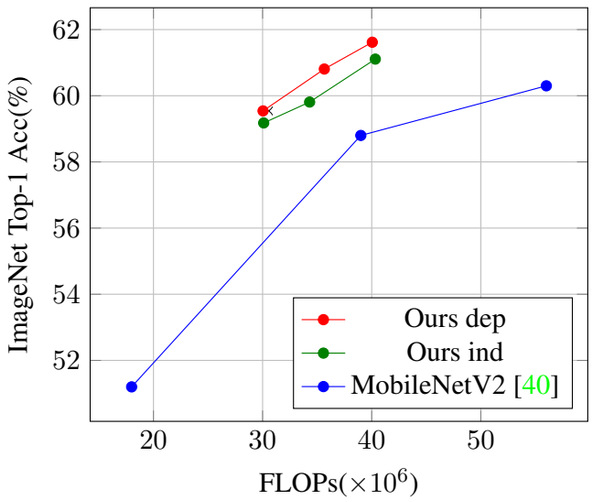
[3]Note that the number we use for their FLOPs is different from what they report. They report lower FLOPs for the baseline ResNet-50 architecture (3.8 GFLOPs versus our 4.028). To normalize the comparison, we added 0.2 GFLOPs to their results.



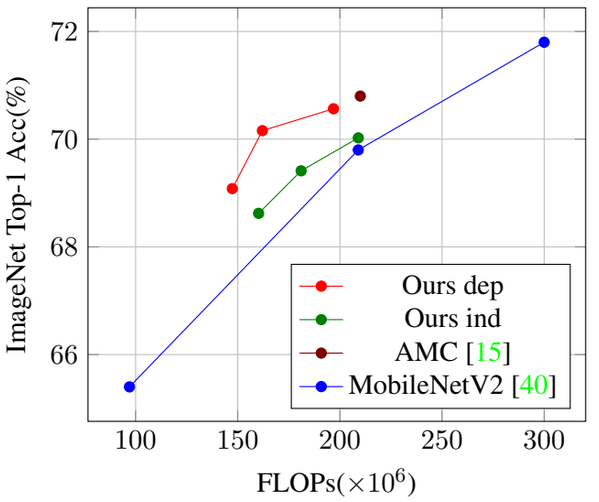
(a) Conditional computation results for ResNet-50



(b) Pruning results for ResNet-50



(c) Pruning and conditional computation results for MobileNetV2 on 96×96. We reduce FLOPs by 37% as shown by the arrow.



(d) Pruning and conditional computation results for MobileNetV2 on 224×224

Figure 3: Selected experimental results for ImageNet.

Model	GFLOPs	Top-1 Acc	Top-5 Acc
ResNet-50	4.03	76.13	92.88
AIG 50 $t = 0.7$	3.26	76.18	92.92
AIG 50 $t = 0.6$	3.08	75.78	92.79
AIG 50 $t = 0.5$	2.91	75.58	92.58
AIG 50 $t = 0.4$	2.76	75.25	92.39
ANN at tradeoff 1	2.7	74.9	91.8
ANN at tradeoff 2	2.6	74	91.8
ANN at tradeoff 3	1.9	71	91.7
MSDNet-3.1	3.1	75.8	-
MSDNet-3	3	75	-
MSDNet-2.1	2.1	74	-
MSDNet-1	1	71	-
Ours dep $t = .5$	2.21	76.3	93.01
Ours dep $t = .4$	1.94	76.04	92.79
Ours dep $t = .3$	1.67	75.19	92.50

(a) Conditional computation for ResNet-50

Model	GFLOPs	Top-1 Acc	Top-5 Acc
ResNet-50	4.03	76.13	92.88
DDS-41	3.47	75.44	90.79
DDS-32	2.82	74.18	91.91
DDS-26	2.33	71.82	92.61
ThiNet-70	2.82	72.05	90.02
ThiNet-50	2.01	71.02	90.67
AutoPruner $r = 0.5$	2.08	74.76	92.15
AutoPruner $r = 0.3$	1.52	73.05	91.25
Ours ind $t = .5$	2.51	76.2	93.08
Ours ind $t = .4$	2.28	76.14	92.92
Ours ind $t = .3$	1.97	75.56	92.52

(b) Pruning for ResNet-50

Model	MFLOPs	Top-1 Acc	Top-5 Acc
Mv2-96 $w = 1.0$	56	60.3	83.2
Mv2-96 $w = 0.75$	39	58.8	81.6
Mv2-96 $w = 0.5$	18	51.2	75.8
Ours ind $t = .5$	40.33	61.11	83.02
Ours ind $t = .3$	34.31	59.81	82.28
Ours ind $t = .1$	30.1	59.18	81.38
Ours dep $t = .5$	40.05	61.62	83.49
Ours dep $t = .3$	35.66	60.81	82.63
Ours dep $t = .1$	30.04	59.54	81.78

(c) MobileNetV2 96 $\times$ 96

Model	MFLOPs	Top-1 Acc	Top-5 Acc
Mv2-224 $w = 1.0$	300	71.8	91.0
Mv2-224 $w = 0.75$	209	69.8	89.6
Mv2-224 $w = 0.5$	97	65.4	86.4
AMC	210	70.8	-
Ours ind $t = .5$	209.02	70.02	89.44
Ours ind $t = .3$	181.01	69.41	88.98
Ours ind $t = .1$	160.21	68.62	88.52
Ours dep $t = .5$	196.85	70.56	89.62
Ours dep $t = .3$	162.07	70.16	89.48
Ours dep $t = .1$	147.34	69.08	88.85

(d) MobileNetV2 224 $\times$ 224

Figure 4: Tabulated experimental results.

Variant	FLOP %	Prec@1	Prec@1 Δ
Conditional computation on ResNet110			
AIG-110 $t = .8$	82%	93.39 $\rightarrow$ 94.24	1% $\uparrow$
Ours dep $t = .6$	65%	93.39 $\rightarrow$ 94.36	1% $\uparrow$
Pruning on ResNet56			
AMC	50%	92.8 $\rightarrow$ 91.9	1.0% $\downarrow$
Ours ind $t = .5$	50%	93.86 $\rightarrow$ 93.31	0.4% $\downarrow$

Figure 5: CIFAR-10 results. The FLOPs is reported as a percentage of the original model and accuracy is reported as baseline $\rightarrow$ new. Note that our ResNet56 baseline has much higher accuracy than AMC's ResNet56 baseline.

Model	FLOPs (10^9)	FLOPs Δ	Prec@1
AIG-50 $t = 0.6$	3.08	76.5%	75.78
Ours layer dep $t = 0.5$	2.72	67.5%	75.78
Ours filter dep $t = 0.4$	1.94	48.1%	76.07

Figure 6: Layer vs Filter granularity for gating. FLOPs Δ is calculated from baseline ResNet50 architecture.

	Block 1	Block 2	Final Block
Images classified	28.71%	11.56%	59.73%
Acc (all images)	81.36	93.35	94.19
Acc (chosen images)	96.37	98.53	92.63

Figure 7: DenseNet on CIFAR-10 with early exit. For early classifiers, the accuracy on chosen images is higher than on all images. This suggests that the gates are learning to recognize "easy" examples in the first two blocks.

4.5 Analysis and ablation studies

4.5.1 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 6. For pruning (data-independent gates), moving to filter granularity from layers resulted in a roughly 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 [50] 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.

4.5.2 Using SGD on MobileNetV2 input size 96

For MobileNetV2 on size 96, we observed that training the clean model from scratch with SGD lead to higher accuracy than the published results, by about 2 Prec@1. As a result, we completed runs with this training schedule at all published widths and report these numbers in the supplemental material appendix. Note, we did not observe this behavior for size 224. Both our conditional computation and pruning results perform better than the adjusted baseline.

4.5.3 Measuring gate polarization

We experimentally investigated the prevalence of gate polarization. For a typical run on MobileNetV2. 96% of them are greater than 0.95 or less than 0.05. Additional data is included in the supplemental material.

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 sufficiently certain of the answer. We are motivated by MSDNet [18], which investigated any-time classification. We explored early exit on both ResNet and DenseNet; however, consistent with [18], we found that ResNet tended to degrade with intermediate classifiers while DenseNet did not.

Probabilistic gates can be used for early exit in DenseNet; following [49] we place gates and intermediate classifiers at the end of each dense block. At each gate, the network makes a discrete decision as to whether the instance can be successfully classified at that stage. The advantage of using Gumbel here is that the early exit can be trained in an end-to-end fashion unlike [49] which uses reinforcement learning.

For the loss, we use a piece-wise function with a quadratic before the target rate and a constant 0 after; this matches the intuition that we should not penalize the network if it can increase the number of early exists without affecting accuracy. These early exit gates can make good decisions regarding which instances to classify early: the intermediate classifiers have higher accuracy on the instances chosen by the gates. Results are shown in Figure 7.

The network had an overall accuracy of 94.39 while using on average only 68.4% of the layers; our implementation of the original DenseNet architecture achieves an accuracy of 95.24 ([20] reports 95.49). More than a third of examples exited early, while overall error was still low.

Acknowledgments

We are grateful to Andreas Veit who spent considerable effort helping us understand his work on AIG. We also thank Serge Belongie for helpful conversations. 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.

Supplemental Material Appendix

S1. Details and Extensions

In this section, we describe details regarding and extensions to the method described in the main paper.

S1.1. Annealing

In the training stage, we propose annealing the target rate. In particular, we use a step-wise annealing target rate which is decreased by a every k epochs. Typical values are $a = .05$ and $k = 5$. The intuition behind annealing is it prevents the network from too aggressively killing off parts of the network early on. Instead, filters which perform worse in the beginning have a chance to change their representation. In practice, we have observed that over time, activations are not always monotonic and some channels will initially start to be less active but will recover. We observe this behavior more with an annealing schedule than for a fixed target rate.

Additionally, on a conceptual level, we argue that annealing allows the classification loss to provide a stronger signal in the case of low target rates. For a normal run with a low target rate, we observe that the network tends to close gates iteratively. For example for a target rate of $t = 0.1$, in the first 20 batches, about 5 gates will close; in the next 20 batches, another 5 gates will close. However, it is important to consider the difference in classification loss at these two points in time. For the first 20 batches, the classification loss is relatively low, so the network has a relatively strong signal as to which gates contribute the least to the overall network classification. However, the classification loss will naturally increase after the 5 gates are closed. As a result, it's possible that the signal provided by classification loss for the next 5 gates may be weaker or less useful.

S1.2. Const-Quad loss and Variable target rate

The activation loss penalizes the situation where the network can decrease activation while retaining the same accuracy - a scenario which is clearly desirable. So, we propose a piece-wise activation loss as follows: Let $Q = \sum_{0 \leq i < |\mathcal{G}|, 0 \leq j < |\mathcal{B}|} Z_{i,j}$ be the overall activation and and t be the target activation rate. For the per-batch setup, this "const-quad" loss is as follows.

$$\mathcal{L}_{CQ} = \begin{cases} 0 & \text{for } Q \leq t \\ (t - Q)^2 & \text{for } Q > t \end{cases} \tag{3}$$

Additionally, there are many cases where a target activation is not clear and the user simply wants an accurate network with reduced train time. For this training schedule, we propose Variable Target Rates, which treat the target rate as a moving average of the network utilization.

For each epoch, the target rate starts at a specific hyperparameter (to prevent collapse to 1) and then is allowed to change according to the batch's activation rate. The two simplest possibilities for the update step are: 1) moving average, and 2) exponential moving average.

S1.3. Gate Possibilities

We also explored several different ways to apply gates to network structures. Note that for data-independent gates, the gating structure defines the allowed network structures and thus greatly impacts the search space that SGD considers. For data-dependent gates, the gating structure has an implicit tradeoff; namely, smaller gates cost less in terms of FLOPs but have worse predictive capabilities.

We define "locked" gates as the scenario where a single gate affects multiple, different layers. In particular, we explored the scenario where, for a sequence of identical layer, a single gate controlled all corresponding filters for every layer in the sequence (where corresponding indicates the filters have the same size and location in the each identical layer). This allows us to explore slimming the network, where each sequence in the final model still contains identical layers; these layers have just been reduced in size in an identical manner.

We also explore "memory" for gates where all the gates in a layer receives the tentative decision of all the other gates in the layer. In practice, we took the output of all gates in their current form and passed them through an additional sequential of the form: linear layer, batch norm, relu, and a linear layer.

We also note that we can apply these gates to input and output connections by forming a matrix of gate values.

S1.4. FLOPs calculation

We follow the convention set by ResNet [14] and followed by numerous other papers [18, 20, 22, 50]: namely, a single FLOP counts a multiply and an add. Using this convention, we note that ResNet-50 version 1 from [14] has 3.8×10^9 FLOPs. One issue that appears to have caused confusion is the difference between ResNet version 1 (ResNetv1) and ResNet version 1.5 (ResNetv1.5) and ResNet version 2 (ResNetv2). The primary difference between the first two models is the location of the stride. In ResNetv1, the stride happens in the first 1x1 conv in each BottleNeck block. In ResNetv1.5, the stride happens in the 3x3 convolution in each BottleNeck block. The change in stride location results in ResNetv1.5 being more accurate and more computationally expensive. Note that ResNetv1 has 3.8×10^9 FLOPs while ResNetv1.5 and ResNetv2 have 4.028×10^9 FLOPs. The important thing to note is that the default TensorFlow and PyTorch implementations of ResNet for ImageNet use the ResNetv2 configuration, which means that they have 4.028×10^9 FLOPs, not 3.8×10^9. This information can be seen on https://github.com/tensorflow/models/tree/master/official/resnet and confirmed for PyTorch here: https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py#L79.

Model	Gate MFLOPs
ResNet-50	8.51
MobileNetV2 224×224	2.51
MobileNetV2 96×96	1.72

Figure 8: The number of MFLOPs for conditional gates for each model.

We confirmed this with both a FLOPs counter and a manual inspection of the code and calculation of FLOPs. Any paper which directly cites TensorFlow or PyTorch's default repository should have 4.028×10^9 as their FLOPs count for ResNet-50.

Since the FLOPs calculation is an importation part of the algorithm (the number is directly used by the batch activation loss), we have a piece of code that directly calculates FLOPs not including the gates. This is the number we report for data-independent gates, since thresholding removes the need to calculate the gates at inference time. For conditional computation, we use this number plus the cost of gate computation. In previous works, we note that average pooling seem not to be included in the FLOPs calculation. However, we err on the side of caution and include them in the calculation as height times width times number of channels. Then the gate for each layer is composed of one average pool, a 1x1 convolution from input channels to 16 channels repeated num_gates times, and a 1x1 convolution from 16 channels to 2 channels repeated num_gates times. Summing this over all the layers gives a relatively small increase in the number of FLOPs. For example, without the gate computation, the gain in MobileNetv2 for input resolution of 96 is about 40%; with the gate computation, the gain is about 37%. More specifically, for MobileNetV2 96×96, Ours dep $t = .5$ changes from 38.03 MFLOPs to 40.05 MFLOPs. The full gate FLOPs can be seen in Figure 8.

S1.5. Additional Early Exit Details

Here, we focus on the use of probabilistic gates to provide an early exit, when the network is sufficiently certain of the answer. We are motivated by MSDNet [18], which investigated any-time classification. We explored early exit on both ResNet and DenseNet; however, consistent with [18], we found that ResNet tended to degrade with intermediate classifiers while DenseNet did not.

Probabilistic gates can be used for early exit in DenseNet; following [49] we place gates and intermediate classifiers at the end of each dense block. At each gate, the network makes a discrete decision as to whether the instance can be successfully classified at that stage. The advantage of using Gumbel here is that the early exit can be trained in an end-to-end fashion unlike [49] which uses reinforcement learning.

For the loss we use 'const quad' loss (Equation 3). These early exit gates can make good decisions regarding which instances to classify early: the intermediate classifiers have higher accuracy on the instances chosen by the gates. Results are shown in DenseNet figure in the paper.

The network had an overall accuracy of 94.39 while use on average only 68.4% of the layers; our implementation of the original DenseNet architecture achieves an accuracy of 95.24 ([20] reports 95.49). More than a third of examples exited early, while overall error was still low.

For our early exit classifiers, we use the same classifiers as [18]. For the gate structure, we use a stronger version of the gate described by [50]. The gates are comprised of the following: a 3×3 convolutional layer with stride of 1 and padding of 1 which takes the current state of the model and outputs 128 channels, a BatchNorm, another 3×3 convolutional layer with stride of 1 and padding of 1 which outputs 128 channels, a BatchNorm, a 4×4 average pool, a linear layer, and then finally a GumbleSoftmax.

S2. Additional data and experiments

S2.1. Observed Polarization

We observe a strong tendency towards gate probability polarization. For example, in a typical run (pruning on MobileNetV2 at resolution 224), a histogram of the learned gate probabilities is included in Figure 9.

S2.2. Results for different inference techniques

We report the measured FLOPs and accuracy for several inference styles in Figure 11.

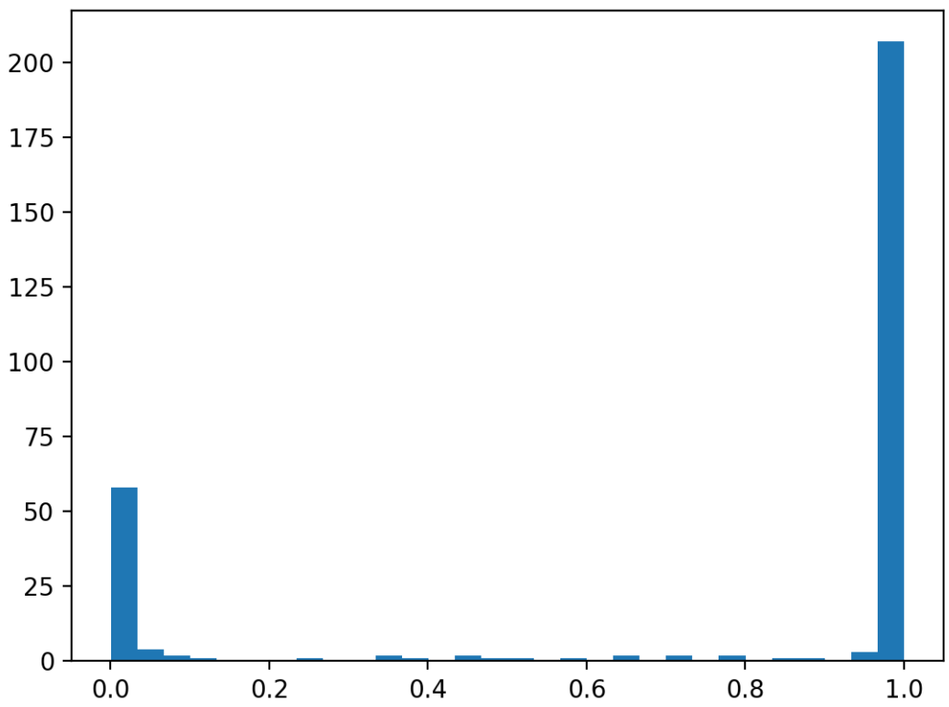


Figure 9: A histogram of learned gate probabilities for a typical pruning run. They demonstrate strong polarization. The x-axis is gate activation rate. The y-axis is the number of gates with that gate activation rate.

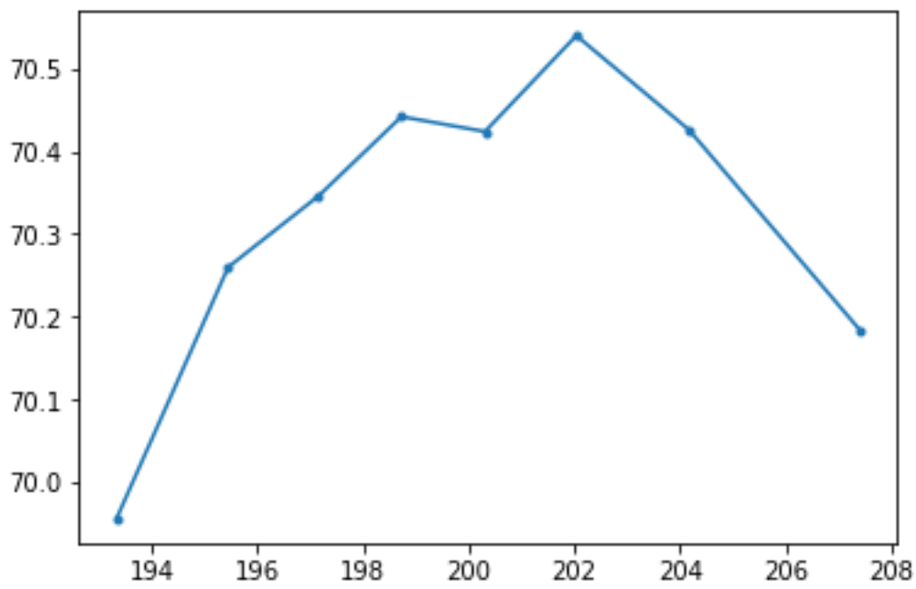


Figure 10: Effect of varying τ, taking on eight values between 0.1 and 0.8. The x-axis is FLOPs (10^6) and the y-axis is Accuracy Prec@1.

Model	MFlops ($\tau = 0.5$)	Prec@1 ($\tau = 0.5$)	MFlops ($\tau = 0.8$)	Prec@1 ($\tau = 0.8$)	Prec@1 (ensemble)	Prec@1 (stoch)
MobilenetV2 Pruning 96 $t = .5$	40.33	61.11	40.27	61.12	61.13	60.73
MobilenetV2 Pruning 96 $t = .3$	34.31	59.81	34.12	59.73	59.81	59.49
MobilenetV2 Pruning 96 $t = .1$	30.1	59.18	30.1	59.17	59.15	58.92
MobilenetV2 Conditional 96 $t = .5$	40.05	61.62	37.01	61.03	61.63	60.96
MobilenetV2 Conditional 96 $t = .3$	35.66	60.81	33.03	60.33	61.01	60.22
MobilenetV2 Conditional 96 $t = .1$	30.04	59.54	27.58	59.22	59.71	59.01
MobilenetV2 Pruning 224 $t = .5$	209.02	70.02	208.87	69.98	70.06	69.69
MobilenetV2 Pruning 224 $t = .3$	181.01	69.41	181.01	69.41	69.45	69.02
MobilenetV2 Pruning 224 $t = .1$	160.21	68.62	160.21	68.62	68.57	68.28
MobilenetV2 Conditional 96 $t = .5$	196.85	70.56	187.56	70.04	70.8	69.95
MobilenetV2 Conditional 96 $t = .3$	162.07	70.16	154.14	69.46	70.3	69.46
MobilenetV2 Conditional 96 $t = .1$	147.34	69.08	140.19	68.57	69.36	68.41
Resnet-50 Conditional $t = .5$	2.21	76.3	2,095.71	75.86	76.39	75.75
Resnet-50 Conditional $t = .4$	1.94	76.04	1,836.57	75.57	76.09	75.43
Resnet-50 Conditional $t = .3$	1.67	75.19	1,587.61	74.82	75.54	74.82
Resnet-50 Pruning $t = .5$	2.51	76.2	2,494.19	76.25	76.15	75.92
Resnet-50 Pruning $t = .4$	2.28	76.14	2,264.99	76.12	76.13	75.8
Resnet-50 Pruning $t = .3$	1.97	75.56	1,935.83	75.52	75.5	75.17

Figure 11: Results on ImageNet for a number of inference styles, including thresholding at two values, an ensemble of 5 different stochastic runs, stochastic (reported accuracies are a mean of 5 runs)

	ResNet Dep t=0.5	ResNet Dep t=0.4	ResNet Dep t=0.3	ResNet Ind t=0.5	ResNet Ind t=0.4	ResNet Ind t=0.3
FLOPs	2.213	1.939	1.673	2.494	2.284	1.969
Prec@1 - Threshold $\tau = 0.5$	76.30	76.04	75.19	76.20	76.14	75.56
Prec@5 - Threshold $\tau = 0.5$	93.01	92.79	92.50	93.08	92.92	92.52
Prec@1 Mean - Stochastic	75.75	75.43	74.82	75.92	75.79	75.17
Prec@1 StdDev - Stochastic	0.029	0.082	0.054	0.057	0.069	0.041
Prec@5 Mean - Stochastic	92.78	92.55	92.23	92.87	92.73	92.34
Prec@5 StdDev - Stochastic	0.035	0.049	0.036	0.067	0.023	0.023
Prec@1 - Ensemble	76.39	76.09	75.54	76.15	76.13	75.50
Prec@5 - Ensemble	93.12	92.88	92.62	93.02	92.93	92.50

Figure 12: A comparison of inference strategies for filter-based pruning on ResNet-50

S2.3. Results from thresholding at different τ values

We explored the thresholding strategy, varying τ for conditional computation on MobileNetV2 with target rate 0.5. The resulting flops/accuracy tradeoff is shown in Figure 10.

S2.4. Detailed results for difference inference techniques on ResNet-50

Table 12 includes the results of running all inference techniques on the ResNet-50 data for the models reported in the paper (using the filter-based granularity). The relation between the inference techniques is consistent across models.

S3. Additional training information for MobileNetV2

S3.1. Input size 96 runs with SGD

In Figure 13, we report the change in performance from the reported numbers in the MobileNetV2 [40] paper and our runs using PyTorch and SGD.

Model	Reported Accuracy $\rightarrow$ Our Accuracy
$w = 1.0$	$60.3 \rightarrow 62.47$
$w = 0.75$	$58.8 \rightarrow 60.94$
$w = 0.5$	$51.2 \rightarrow 51.25$
$w = 0.35$	$45.5 \rightarrow 45.52$

Figure 13: MobileNetv2 reported numbers vs our runs with PyTorch and SGD.

To make the comparison simpler, we show this adjusted baseline in Figure 14.

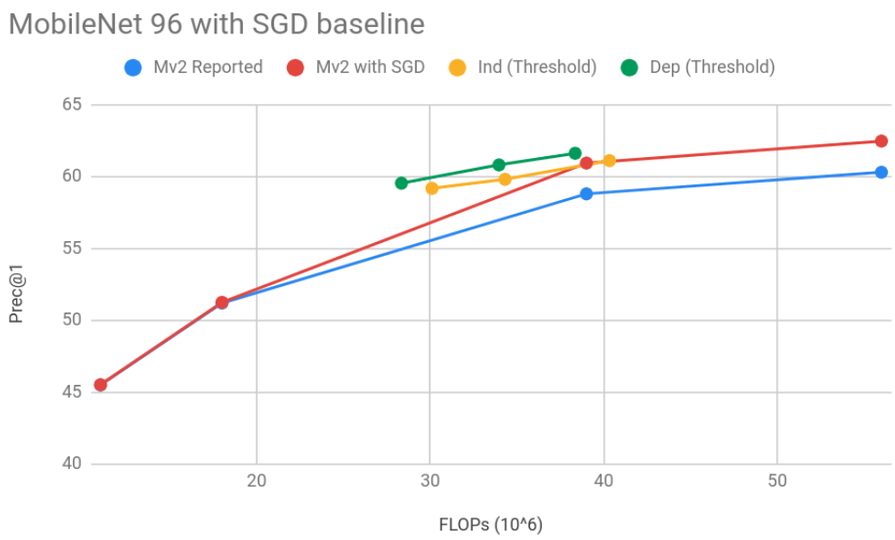


Figure 14: MobileNetV2 96 results with adjusted baseline. Note that even though our gains are not as pronounced, we still improve upon the new baseline by approximately 20%.

S3.2. MobileNetV2 runs with 0.98 decay rate

In Figure 15, we show the results of using batch rate 420, learn rate 0.08, and a 0.98 decay rate for the learning rate over each epoch. Note that for both target rates, there is not much variation between the two schemes; the extra epochs seem to result in slightly higher accuracy at slightly fewer FLOPs.

	FLOPs - Threshold	Prec@1 - Threshold at 0.5	Prec@5 - Threshold at 0.5
Mobile224 Dep $t = 0.1$ decay=0.98	149.547	68.99	88.706
Mobile224 Dep $t = 0.5$ decay=0.98	198.719	70.47	89.77
Mobile224 Dep $t = 0.1$ decay=0.99	144.841	69.082	88.85
Mobile224 Dep $t = 0.5$ decay=0.99	194.352	70.564	89.622

Figure 15: Mobile224 runs with decay rate of 0.99 and 0.98. Note that they are fairly similar.

S4. Layer-based Approach

S4.1. Approach description

We also investigated a layer-pruning approach for ResNet. As in AIG, for a residual layer $f_l()$, we typically have

$$x_{l+1} = x_l + f_l(x_l)$$

which we simply replace by a probabilistic gate to get

$$x_{l+1} = x_l + Z_l(f_l(x_l))$$

so that the layer's being run is dependent on the value of the gate. In the below, we report layer-pruning results; the baseline model is ResNet-50. For brevity we adopt the following abbreviations:

- Loss functions are either **PB** (batch activation loss, our proposed loss) or **PG** (per-gate, as in AIG).
- **Ind** and **Dep** stand for independent (pruning) or dependent (conditional computation) gates.
- **Act** measures the overall activations, i.e., the (average) fraction of the gates that are on

S4.2. Polarization

The following is a summary of polarization on the layer-based granularity.

With the per-batch loss, we often observe polarization, where some layers are nearly always on and some nearly always off. In the case of data-dependent bypass, we can measure the observed activation of a gate during training. For example, on a per-batch run on ResNet-50 (16 gates) on ImageNet, nearly all of the 16 gates polarize, as shown in Figure 16: four gates collapsed to zero or one exactly; more than half were at their mode more than 99.9% of the time. Interestingly, we observe different activation behavior on different datasets. ImageNet leads to frequent and aggressive polarization, all networks exhibited some degree of polarization; CIFAR10 can induce polarization but does so much less frequently, approximately less than 40% of our runs.

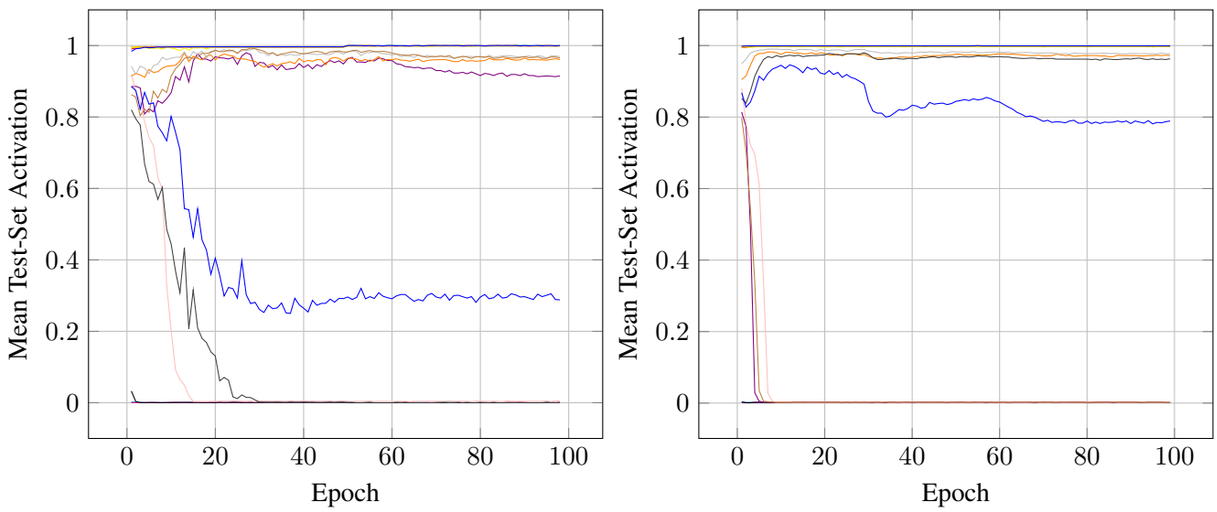


Figure 16: Demonstration of polarization on layer-based granularity on (left) data-dependent, per-batch ResNet-50 on ImageNet with target rate of .5, and (right) data-independent per-batch with target rate of .4 (right). Nearly all of the 16 gates collapse.

S4.3. ImageNet Results

In Figure 18 and Tables 19, 20, 21 and 22 we report all data collected on a layer-based granularity. See Figure 17 for a summary table.

We report all the data collected on ImageNet using the different gate techniques (independent, dependent), target loss techniques (per-batch, per-gate), and inference time techniques (threshold, always-on, stochastic, ensemble). Note that we try to include AIG for reference whenever it's a fair comparison.

Also of note, for the ensemble technique we also include data from [19]. Note that using the stochastic networks, we outperform their ensemble technique. Also note that their technique is orthogonal to ours, so both could be used to identify an even better ensemble.

In general, we observe that unsurprisingly, ensemble has the highest performance in terms of error; however, this requires multiple forward passes through the network, so the performance gain is offset by the inference time required. We also observe that threshold generally outperforms stochastic.

For all ImageNet results, we used the pretrained models provided by TorchVision[4].

[4]The model links are provided here: https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

Model	Top-1 Stochastic	Top-5 Stochastic	GFLOPs Stochastic	Top-1 Det.	Top-5 Det.	# GFLOPs Det.
ResNet-34	-	-	-	26.69	8.58	3.6
ResNet-50	-	-	-	23.87	7.12	3.8
AIG 50 [t=0.4]	24.75	7.61	2.56	-	-	2.56
AIG 50 [t=0.5]	24.42	7.42	2.71	-	-	2.71
AIG 50 [t=0.6]	24.22	7.21	2.88	-	-	2.88
AIG 50 [t=0.7]	23.82	7.08	3.06	-	-	3.06
Ind PG* [t=0.5]	24.99 (0.05)	7.71 (0.04)	3.81	24.23	7.17	3.04
Dep PG* [t=0.4]	25.25 (0.08)	7.81 (0.05)	2.51	24.78	7.57	2.52
Dep PG* [t=0.5]	24.92 (0.05)	7.50 (0.02)	2.73	24.52	7.27	2.79
Dep PG* [t=0.6]	24.47 (0.07)	7.36 (0.05)	2.99	24.07	7.16	3.03
Ind PB* [t=0.4]	24.70 (0.08)	7.63 (0.03)	2.43	24.42	7.48	2.49
Ind PB* [t=0.5]	24.39 (0.05)	7.46 (0.03)	2.65	24.04	7.29	2.71
Ind PB* [t=0.6]	**24.04 (0.03)**	7.11 (0.03)	2.93	**23.72**	**6.93**	2.93
Dep PB* [t=0.4]	24.98 (0.02)	7.63 (0.10)	2.27	24.75	7.56	2.28
Dep PB* [t=0.5]	24.22 (0.04)	7.18 (0.03)	2.52	23.99	7.06	2.55
Dep PB* [t=0.6]	24.16 (0.05)	7.24 (0.01)	2.71	23.99	7.14	2.73
ResNet-101	-	-	-	22.63	6.45	7.6
AIG 101 [t=0.3]	23.02	6.58	4.33	-	-	-
AIG 101 [t=0.4]	22.63	6.26	5.11	-	-	-
Dep PB* [t=0.5]	22.73 (0.02)	6.46 (0.02)	4.48	22.43	6.34	4.60
Dep PB* [t=0.6]	**22.45 (0.08)**	6.28 (0.04)	5.11	**22.22**	6.28	5.28

Figure 17: Error and GFLOPs for our method (marked with asterisks) compared to ConvNet-AIG. For stochastic errors we run the test set 5 times and report mean and, in parenthesis, standard deviation. For deterministic error, we use the thresholding inference technique.

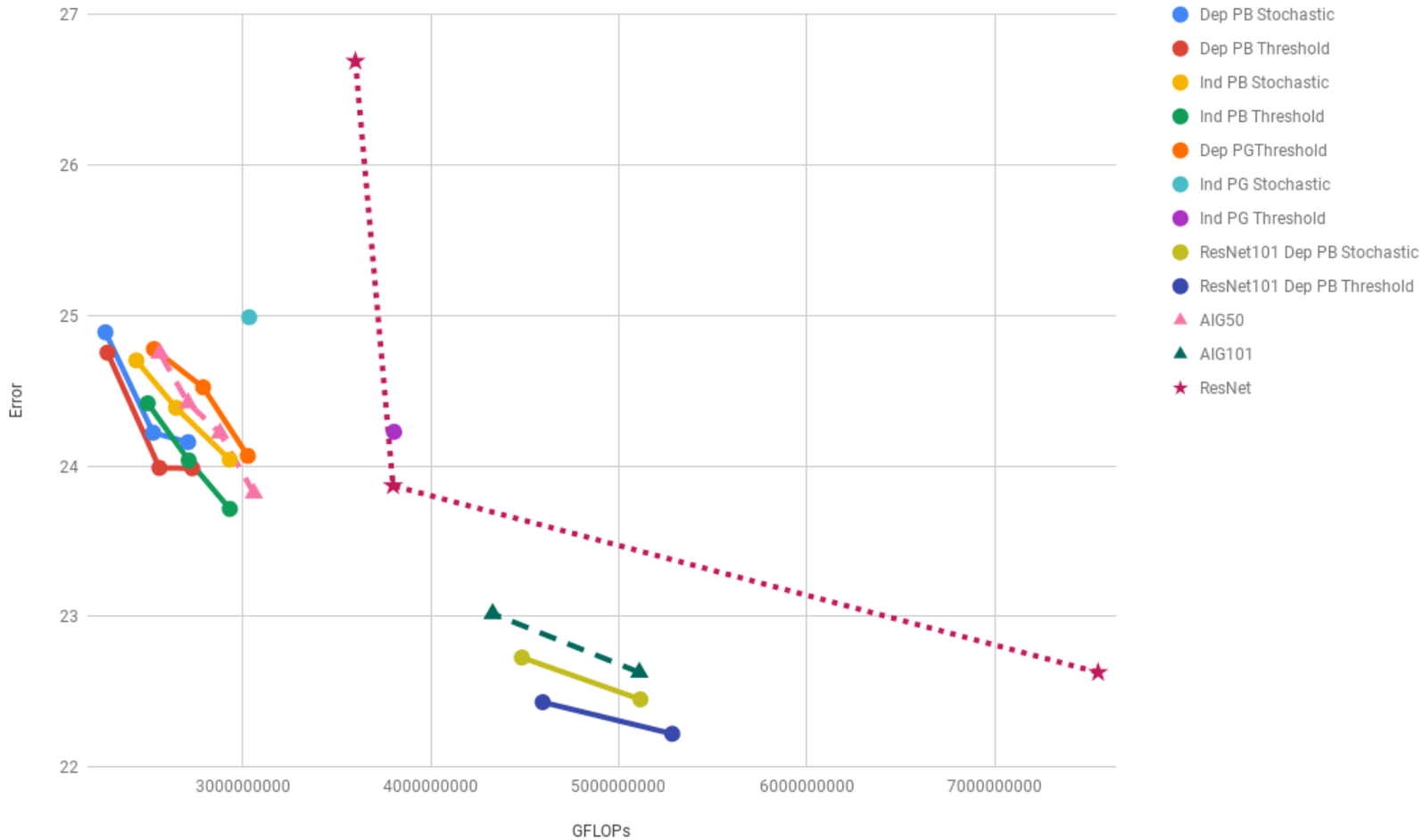


Figure 18: Full comparison of the tradeoff for inference techniques for ResNet. Note the y-axis reports Prec@1 Error.

Technique	Threshold GFlops	Threshold Prec@1	Threshold Prec@5	Threshold Acts
Dep PB @ tr=.6	2732360786	23.986	7.14	0.6929
Dep PB @ tr=.5	2557646188	23.988	7.058	0.6429
Dep PB @ tr=.4	2281151219	24.754	7.562	0.5637
Ind PB @ tr=.6	2932012776	**23.716**	**6.934**	0.75
Ind PB @ tr=.5	2713646824	24.038	7.294	0.6875
Ind PB @ tr=.4	2495280872	24.418	7.478	0.625
Dep PB ConstQuad Anneal	2717097006	23.98	6.982	0.6885
Dep PG @ tr=.6	3027899835	24.068	7.16	0.7726
Dep PG @ tr=.5	2789914344	24.524	7.272	0.69
Dep PG @ tr=.4	2528475066	24.778	7.57	0.6098
Ind PG @ tr=.5	3805476584	24.228	7.168	1
AIG @ tr=.7	3060000000	23.82	7.08	-
AIG @ tr=.6	2880000000	24.22	7.21	-
AIG @ tr=.5	2710000000	24.42	7.41	-
AIG @ tr=.4	2560000000	24.75	7.61	-
Res101 Dep PB @ tr=.6	5284996031	**22.222**	6.28	0.69
Res101 Dep PB @ tr=.5	4596138799	22.432	6.336	0.594
AIG101 @ tr=.5	5110000000	22.62	**6.26**	-
AIG101 @ tr=.3	4330000000	23.02	6.58	-

Figure 19: Full comparison of the tradeoff for inference techniques for ResNet (Thresholding). Note that this table reports Error for Prec@1 and Prec@5.

Technique	AlwaysOn Prec@1	AlwaysOn Prec@5	AlwaysOn Acts
Dep PB @ tr=.6	23.968	7.134	1
Dep PB @ tr=.5	24.362	7.248	1
Dep PB @ tr=.4	25.25	7.826	1
Ind PB @ tr=.6	**23.75**	**6.944**	1
Ind PB @ tr=.5	24.08	7.362	1
Ind PB @ tr=.4	24.46	7.448	1
Dep PB ConstQuad Anneal	23.932	6.994	1
Dep PG @ tr=.6	23.864	6.968	1
Dep PG @ tr=.5	24.406	7.19	1
Dep PG @ tr=.4	24.734	7.536	1
Ind PG @ tr=.5	24.228	7.168	1
ResNet 50	23.87	7.12	1
Res101 Dep PB @ tr=.6	**22.42**	**6.222**	1
Res101 Dep PB @ tr=.5	23.276	6.708	1
ResNet 101	22.63	6.45	

Figure 20: Full comparison of the tradeoff for inference techniques for ResNet (Always-run-gates). Note that this table reports Error for Prec@1 and Prec@5.

Stochastic Technique	GFlops	Prec@1 Mean	Prec@1 StdDev	Prec@5 Mean	Prec@5 StdDev	Acts Mean	Acts StdDev
Dep PB @ tr=.6	2710541198	24.1592	0.047	7.24	0.0055	0.6862	0.0002
Dep PB @ tr=.5	2524273294	24.222	0.0358	7.18	0.0278	0.6321	0.0002
Dep PB @ tr=.4	2270347346	24.8892	0.0224	7.6284	0.096	0.5596	0.0001
Ind PB @ tr=.6	2932012776	24.0428	0.0246	7.1056	0.0282	0.7222	0.0001
Ind PB @ tr=.5	2646139684	24.3876	0.0496	7.46	0.0299	0.6676	0.0001
Ind PB @ tr=.4	2434652927	24.7024	0.0753	7.634	0.0291	0.6068	0.0002
Dep PB ConstQuad Anneal	2638223224	24.4576	0.0753	7.2944	0.0403	0.66631	0.0003
Dep PG @ tr=.6	2988683827	24.472	0.0741	7.3616	0.0508	0.7549	0.0002
Dep PG @ tr=.5	2734859869	24.9176	0.052	7.4972	0.0231	0.6797	0.0002
Dep PG @ tr=.4	2510905649	25.2476	0.0802	7.8084	0.047	0.6136	0.0002
Ind PG @ tr=.5	3035128056	24.9892	0.05	7.7064	0.0345	0.7681	0.0005
AIG @ tr=.7	3060000000	23.82	-	7.08	-	-	-
AIG @ tr=.6	2880000000	24.22	-	7.21	-	-	-
AIG @ tr=.5	2710000000	24.42	-	7.41	-	-	-
AIG @ tr=.4	2560000000	24.75	-	7.61	-	-	-
Res101 Dep PB @ tr=.6	5115483937	**22.4508**	0.077	6.2768	0.0346	0.6665	0
Res101 Dep PB @ tr=.5	4485480286	22.7288	0.0229	6.4552	0.0206	0.5791	0.0001
AIG101 @ tr=.5	5110000000	22.62		**6.26**			
AIG101 @ tr=.3	4330000000	23.02		6.58			

Figure 21: Full comparison of the tradeoff for inference techniques for ResNet (Stochastic). Note that this table reports Error for Prec@1 and Prec@5.

Technique	Ensemble Prec@1	Ensemble Prec@5
Dep PB @ tr=.6	23.952	7.14
Dep PB @ tr=.5	24.014	7.076
Dep PB @ tr=.4	24.76	7.558
Ind PB @ tr=.6	**23.814**	**6.97**
Ind PB @ tr=.5	24.116	7.32
Ind PB @ tr=.4	24.448	7.506
Dep PB ConstQuad Anneal	23.952	7.09
Dep PG @ tr=.6	23.844	7.1
Dep PG @ tr=.5	24.342	7.188
Dep PG @ tr=.4	24.62	7.486
Ind PG @ tr=.5	24.376	7.328
ResNet 50	23.87	7.12
Snapshot [19]	23.96[5]	
Res101 Dep PB @ tr=.6	**22.186**	**6.162**
Res101 Dep PB @ tr=.5	22.42	6.31
ResNet 101	22.63	6.45

Figure 22: Full comparison of the tradeoff for inference techniques for ResNet (Ensembles). Note that this table reports Error for Prec@1 and Prec@5.

S4.4. CIFAR10 Performance

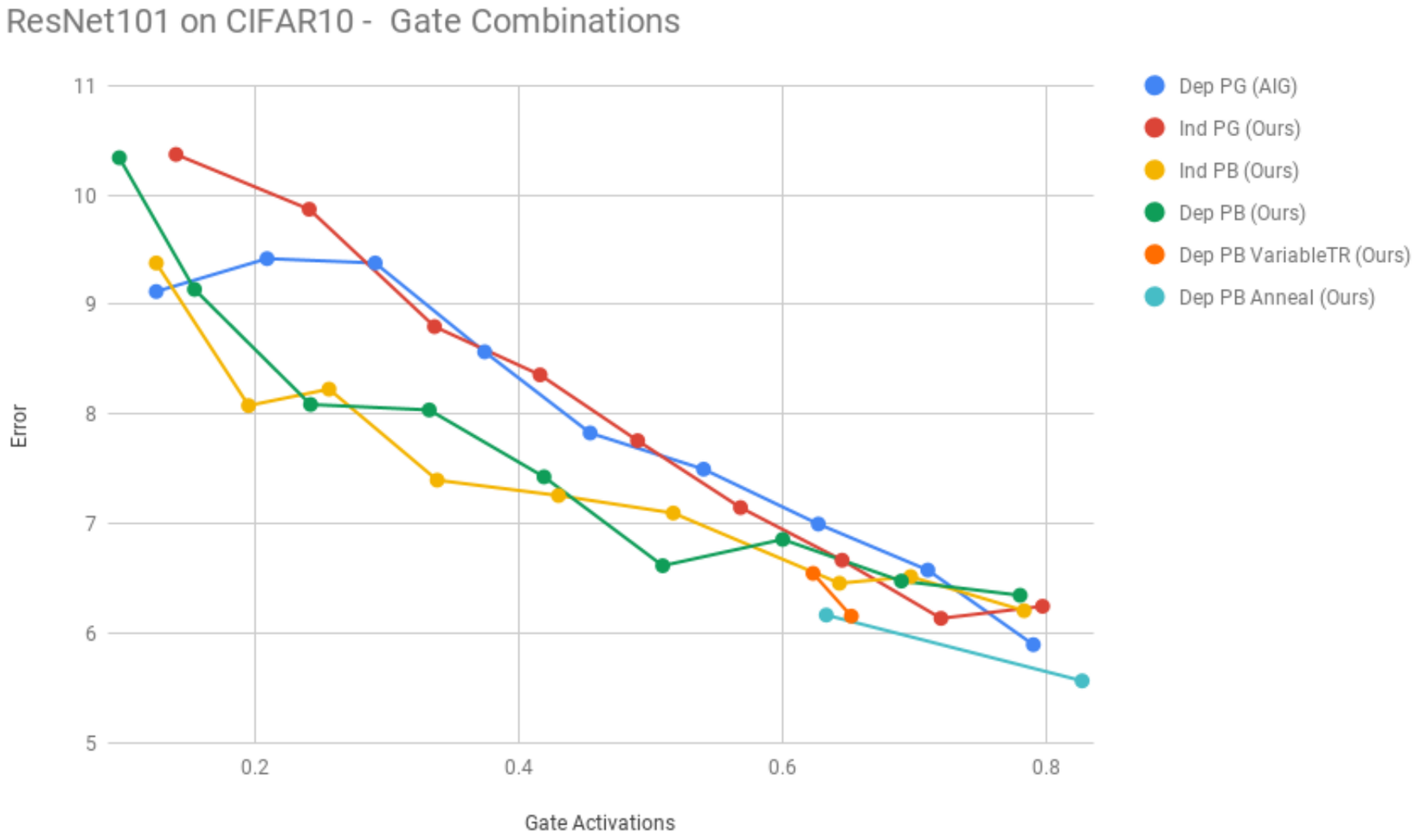
We report all the data collected on CIFAR10 using the different gate techniques (independent, dependent), target loss techniques (per-batch, per-gate). We report only the numbers for the stochastic inference time technique. We used CIFAR10 as a faster way to explore the space of parameters and combinations and as such have a more dense sweep of the combination and parameters. Note that for CIFAR10, we did not use a pretrained model; the entire model is trained from scratch.

In general, we found that for a wide set of parameters per-batch outperforms per-gate. This includes independent per-batch outperforming dependent per-gate.

The only exception to this is very high and very low target rates. However we note that at very high target rates, the accuracy of per-batch can be recovered through annealing. We attribute this to the fact that for CIFAR10 we train from scratch. Since the model is completely blank for the first several epochs, the per-batch loss can lower activations for any layers while still improving accuracy. In other words, at the beginning, the model is so inaccurate that any training on any subset of the model will result in a gain of accuracy; so when training from scratch, the per-batch loss will choose the layers to decrease activations for greedily and sub-optimally.

One surprising result is that independent per-gate works at all for a wide range of target rates. This suggests that the redundancy effect described in [21] is so strong that the gates can be kept during inference time. This also suggests that at least for CIFAR10, most of the gains described in [50] were from regularization and not from specialization.

We also report some variable rate target rates. We note that these tend to outperform the quadratic loss on a constant target rate. We believe that this is because variable target rate allows the optimizer to take the easiest and farther path down the manifold. We note that some of the variable target rates that worked on CIFAR10 did not work on ImageNet; namely, variable target rates which updated the target to previous mean quickly (within 5 epochs) lead to polarization for all gates. We attribute this to the much larger amount of training data for ImageNet and increased complexity of the task. However, both annealing and variable target rates merit further experimentation and research to truly understand how they perform on different datasets and with different training setups (from scratch vs from pretrained).


ResNet101 on CIFAR10 - Gate Combinations
Error
Gate Activations
11
10
9
8
7
6
5
0.2
0.4
0.6
0.8
Dep PG (AIG)
Ind PG (Ours)
Ind PB (Ours)
Dep PB (Ours)
Dep PB VariableTR (Ours)
Dep PB Anneal (Ours)

S4.5. CIFAR10 Activation Graphs

In Figures 24, 25, 26 and 27 we provide graphs of the activation rates for each layer over time. This demonstrates that on CIFAR10, each layer does not instantly polarize to its eventual activation rate; the activation rates can change throughout training time.

Technique	Best Error	Final Error	Activations
Dep PG @tr=.0	9.12	9.26	0.125
Dep PG @tr=.1	9.42	9.8	0.209
Dep PG @tr=.2	9.38	9.4	0.291
Dep PG @tr=.3	8.57	8.88	0.374
Dep PG @tr=.4	7.83	8.3	0.454
Dep PG @tr=.5	7.5	7.5	0.54
Dep PG @tr=.6	7	7.14	0.627
Dep PG @tr=.7	6.58	6.82	0.71
Dep PG @tr=.8	5.9	6.08	0.79
Dep PB @tr=.0	10.34	10.5	0.097
Dep PB @tr=.1	9.14	9.36	0.154
Dep PB @tr=.2	8.09	8.45	0.242
Dep PB @tr=.3	8.04	8.48	0.332
Dep PB @tr=.4	7.43	7.8	0.419
Dep PB @tr=.5	6.62	7.04	0.509
Dep PB @tr=.6	6.86	7.86	0.6
Dep PB @tr=.7	6.48	6.83	0.69
Dep PB @tr=.8	6.35	6.35	0.78
Ind PG @tr=.0	10.37	10.75	0.14
Ind PG @tr=.1	9.87	10.27	0.241
Ind PG @tr=.2	8.8	9.15	0.336
Ind PG @tr=.3	8.36	8.75	0.416
Ind PG @tr=.4	7.76	7.95	0.49
Ind PG @tr=.5	7.15	7.43	0.568
Ind PG @tr=.6	6.67	6.98	0.645
Ind PG @tr=.7	6.14	6.54	0.72
Ind PG @tr=.8	6.25	6.82	0.797
Ind PB @tr=.0	9.38	9.52	0.125
Ind PB @tr=.1	8.08	8.43	0.195
Ind PB @tr=.2	8.23	8.41	0.256
Ind PB @tr=.3	7.4	7.65	0.338
Ind PB @tr=.4	7.26	7.71	0.43
Ind PB @tr=.5	7.1	7.39	0.517
Ind PB @tr=.6	6.46	6.92	0.643
Ind PB @tr=.7	6.52	6.86	0.697
Ind PB @tr=.8	6.21	6.23	0.783
Dep PB VariableTarget Quad @ tr=.8	6.16	6.43	0.652
Dep PB VariableTarget ConstQuad @ tr=.8	6.55	6.72	0.623
Dep PB Anneal $\Rightarrow$ 1.0 .95 .9 .85 .8	5.57	6	0.827
Dep PB Anneal $\Rightarrow$ .8 .7 .6	6.17	6.66	0.633

Figure 23: Full list of CIFAR-10 results for ResNet-110 at various activations. Note that this table reports Error.

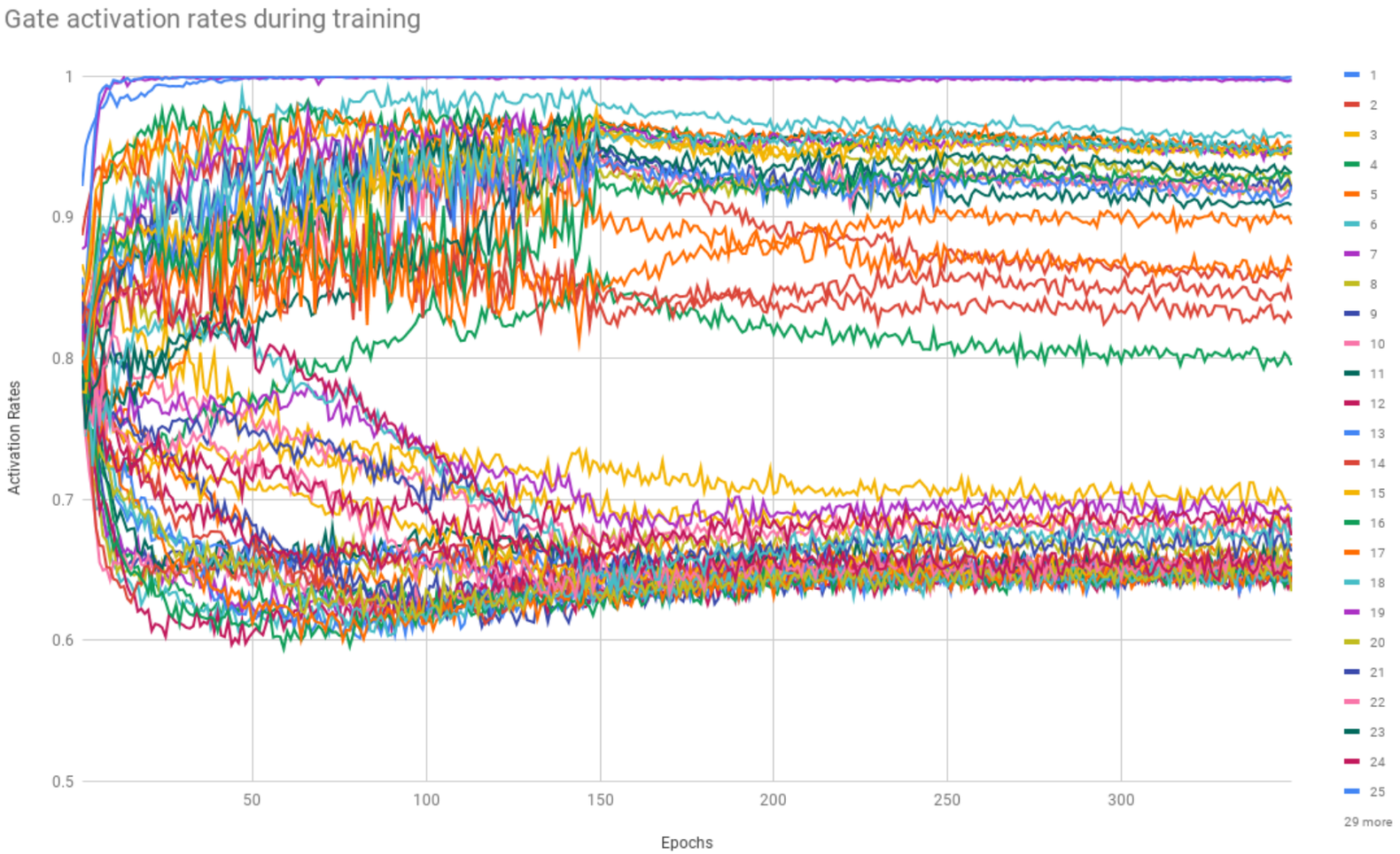


Figure 24: Layer-based ResNet-110 PB with target rate of 0.8

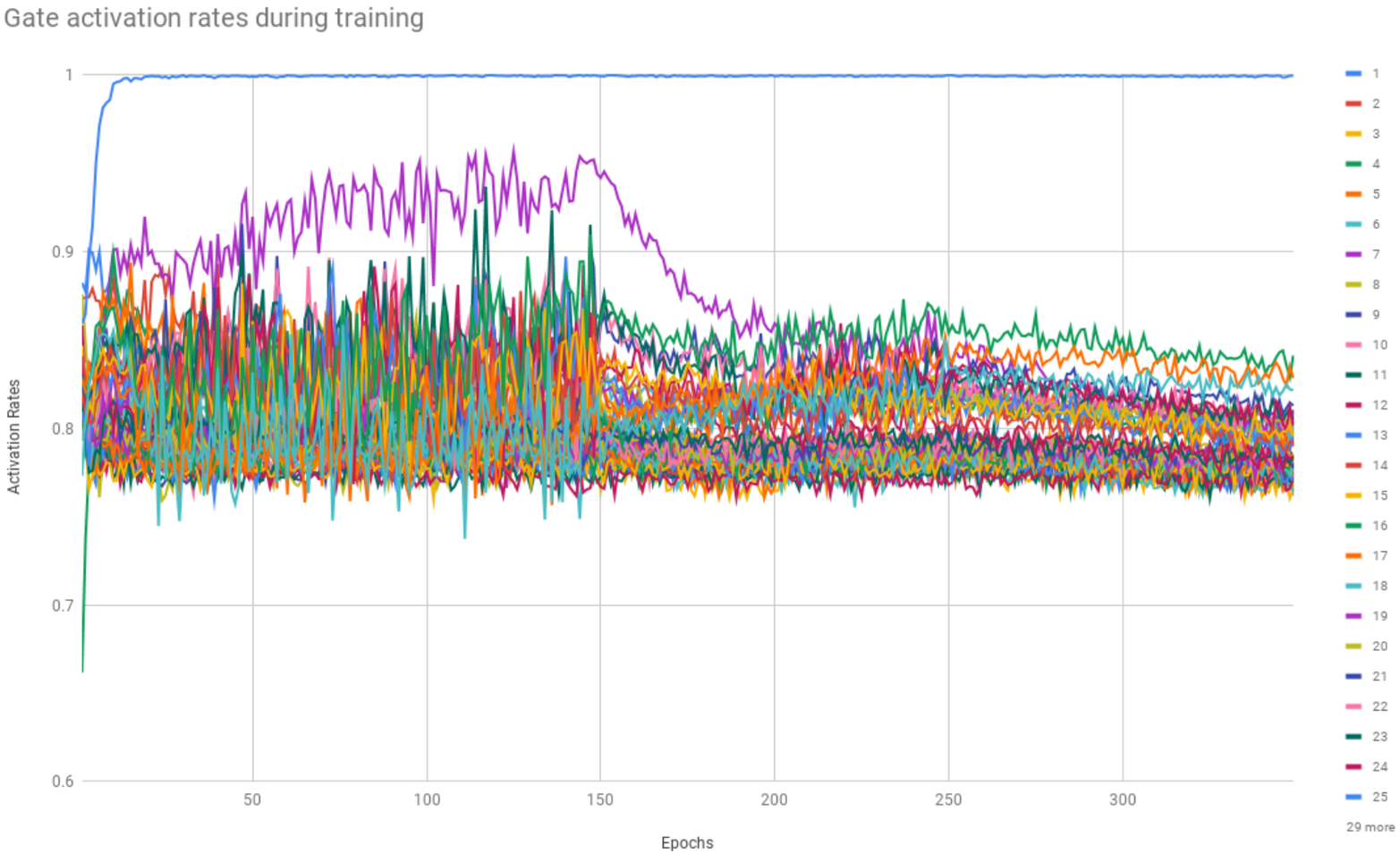


Figure 25: Layer-based ResNet-110 PG with target rate of 0.8

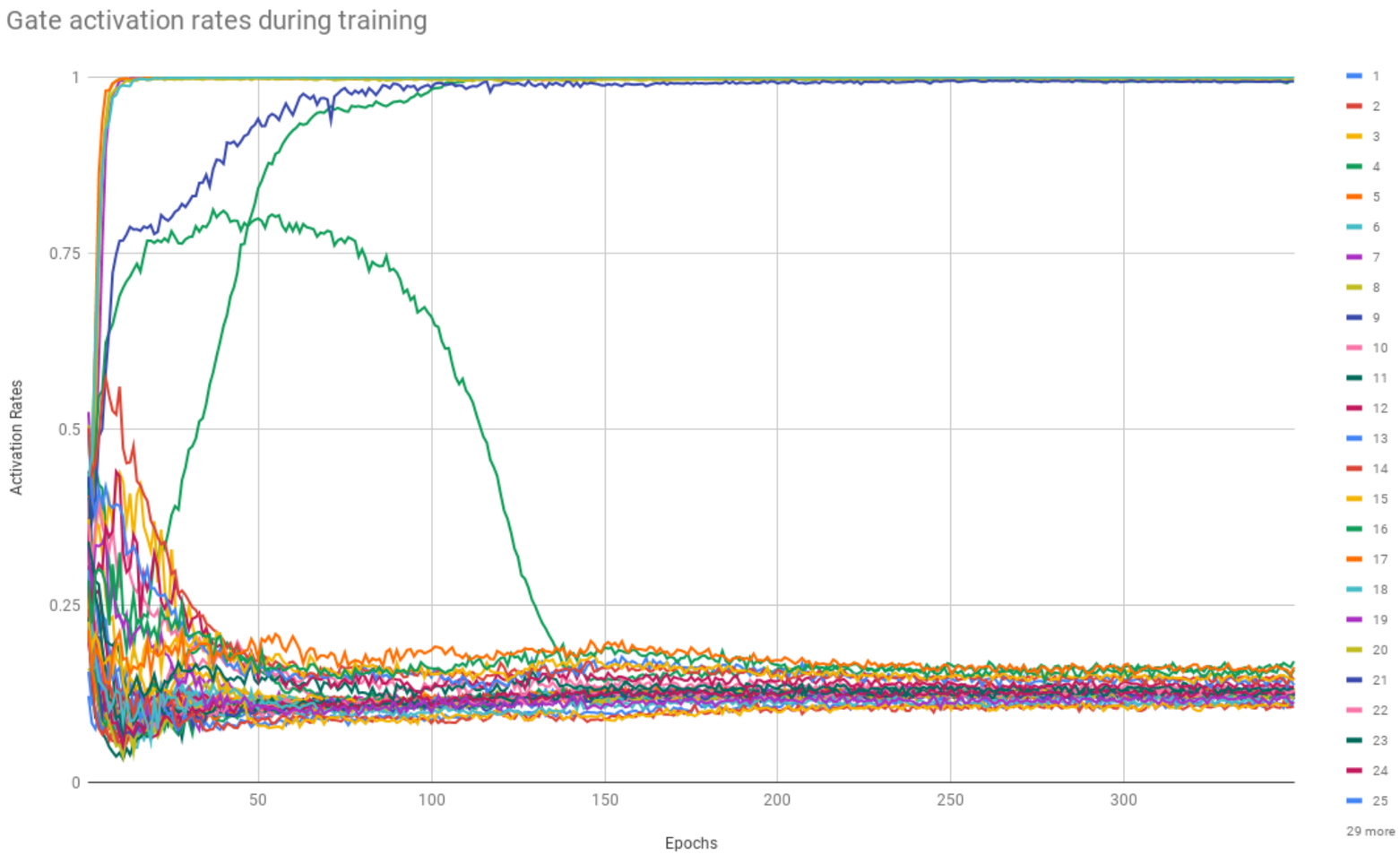


Figure 26: Layer-based ResNet-110 PB with target rate of 0.2

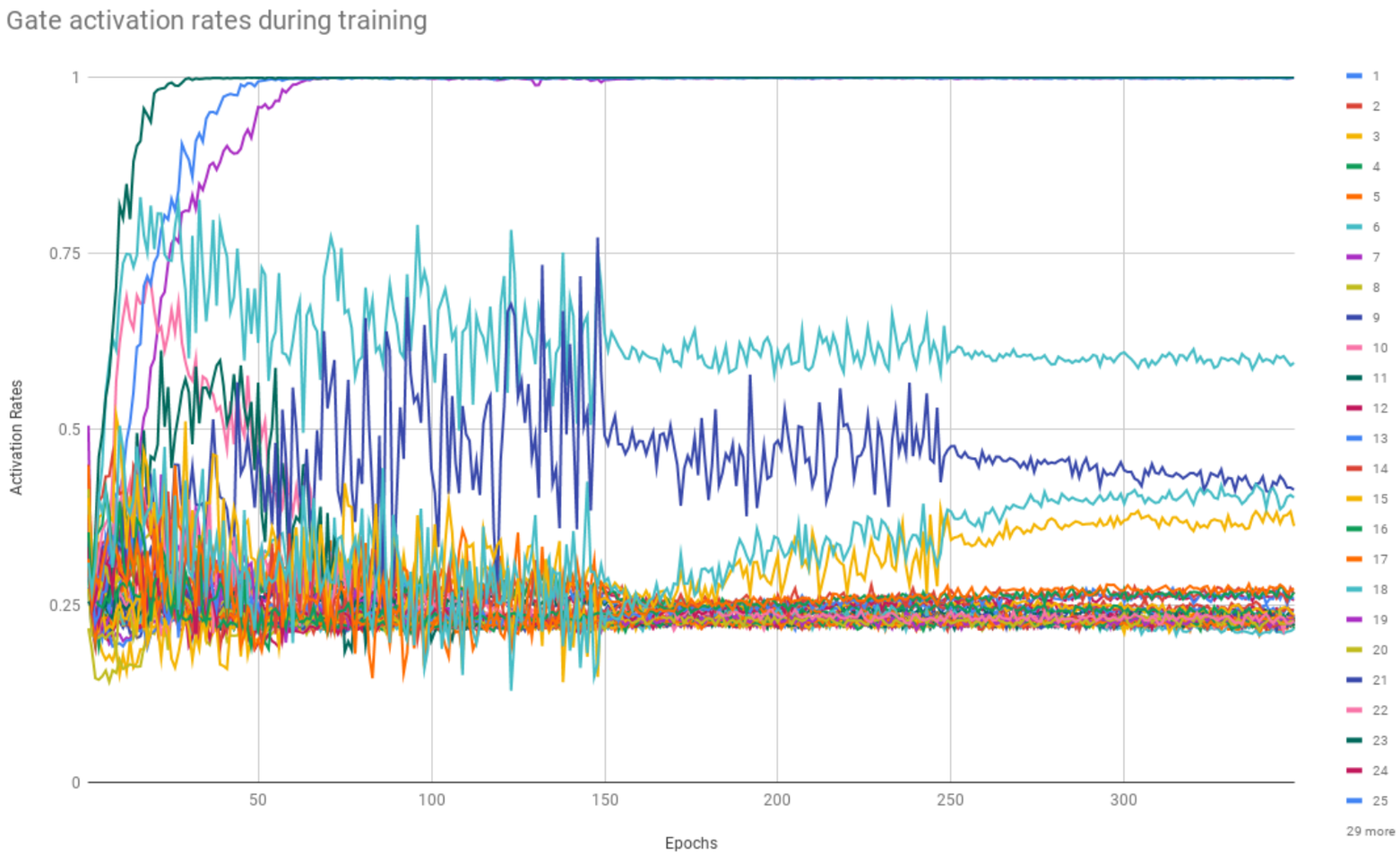


Figure 27: Layer-based ResNet-110 PG with target rate of 0.2

S4.6. Observations regarding gating, classification accuracy, and polarization

Observation 4.1 *A layer with activation of p can only affect the final accuracy by p.*

We use this observation to suggest that polarization is beneficial for classification loss.

In this paragraph, we provide the high level, conceptual argument and will follow it with a more precise, concrete example. Consider a network with only two layers and the restriction that, on expectation, only one layer should be on. Then let p be the probability that layer 1 is on. Intuitively, if $p \notin \{0, 1\}$, then we are in a high entropy state where the network must deal with a large amount of uncertainty regarding which layers will be active. Furthermore, some percentage of the time no layer will be run at inference time, causing the network to completely fail. Additionally, the percentage of the time that the network tries to train both layers to work together may be "wasted" in some sense, since they will only execute together a small percentage of time during inference.

A more concrete example follows:

Observation 4.2 *Consider a network with two layers with data-independent probability p_1 and p_2 of being on, restricted to the case that $p_1 + p_2 = 1$. Let a_1 be the expected accuracy of a one-layer network and a_2 be the expected accuracy of a two-layer network. A polarized network ($p_1 \in \{0, 1\}$) will have higher expected accuracy than a not-polarized one if and only if $\frac{a_2}{a_1} \geq 2$.*

Because of the restriction $p_1 + p_2 = 1$, there is only one parameter for the probabilities. Let $p = p_1$. Then $p(1-p)$ is the probability that both layers will be on and also the probability that both layers will be off. Note that the network has a strict upper bound on accuracy of $1 - p(1-p)$ since with probability $p(1-p)$ none of the layers will activate and no output will be given.

Then the expected accuracy of the network for any probability $p \in [0, 1]$ is $(1 - 2p + 2p^2)a_1 + p(1-p)a_2$, note that for $p \in \{0, 1\}$ the accuracy is simply a_1. For a value $p \in (0, 1)$ to have higher expected accuracy, we need

$$\begin{aligned} a_1 &< (1 - 2p + 2p^2)a_1 + p(1-p)a_2 \\ \frac{-2p^2 + 2p}{p(1-p)} &< \frac{a_2}{a_1} \\ 2 &< \frac{a_2}{a_1} \end{aligned}$$

Note that a strong restriction in this case is identical to the coefficient of the batch activation loss being infinite. As the coefficient decreases, the network gains more flexibility and can trade batch activation loss for classification loss, so the argument does not strictly hold in the case described in the paper. However, we believe that the general intuition and statements still apply.

References

[1] Y. Bengio. Deep learning of representations: Looking forward. *CoRR*, abs/1305.0445, 2013. 1

[2] Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. 1

[3] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for efficient inference. In *ICML*, pages 527–536, 2017. 2, 7

[4] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv:1812.00332*, 2018. 3

[5] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015. 1

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009. 5

[7] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017. 1, 2

[8] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017. 2, 3, 4

[9] A. Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016. 2

[10] E. J. Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series*, 33, 1954. 1

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1, 2

[12] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 2

[13] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993. 2

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 5, 7, 11

[15] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. AMC: Automl for model compression and acceleration on mobile devices. In *ECCV*, pages 784–800, 2018. 1, 7

[16] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 2

[17] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 2

[18] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *CoRR, abs/1703.09844*, 2, 2017. 2, 7, 9, 11, 12

[19] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017. 16, 21

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 9, 11, 12

[21] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, pages 646–661. Springer, 2016. 1, 3, 21

[22] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. *ECCV*, 2018. 1, 2, 7, 11

[23] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017. arXiv preprint arXiv:1611.01144. 1, 2, 3, 4

[24] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NIPS*, pages 2575–2583. Curran Associates, Inc., 2015. 3

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1

[26] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, May 2015. 1

[27] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 1, 2

[28] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2017. 1, 2

[29] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *CVPR*, pages 5325–5334, 2015. 2

[30] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *ICLR*, 2019. 2

[31] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *ICLR*, 2017. arXiv preprint arXiv:1712.01312. 3

[32] J.-H. Luo and J. Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018. 2, 6, 7

[33] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *ICCV*, 2017. arXiv preprint arXiv:1707.06342. 2, 7

[34] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 1

[35] M. McGill and P. Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *ICML*, 2017. arXiv preprint arXiv:1703.06217. 2

[36] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In D. Precup and Y. W. Teh, editors, *ICML*, volume 70, pages 2498–2507. PMLR, 06–11 Aug 2017. 3

[37] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2016. arXiv preprint arXiv:1611.06440. 2

[38] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989. 1, 2

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 5

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018. 1, 5, 7, 15

[41] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. 2

[42] S. Shirakawa, Y. Iwata, and Y. Akimoto. Dynamic optimization of neural network structures using probabilistic modeling. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3

[43] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[44] S. Srinivas and R. V. Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016. 3

[45] S. Srinivas and V. Babu. Learning neural network architectures using backpropagation. In *BMVC*, pages 104.1–104.11, September 2016. 1, 3

[46] S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *CVPR Workshops*, July 2017. 3

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 3

[48] X. Suau, L. Zappella, V. Palakkode, and N. Apostoloff. Principal filter analysis for guided network compression. *arXiv preprint arXiv:1807.10585*, 2018. 2

[49] S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR*, pages 2464–2469. IEEE, 2016. 2, 9, 12

[50] A. Veit and S. Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2017. 1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 21

[51] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 2

[52] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *CVPR*, pages 2129–2137, 2016. 2

[53] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 2

[54] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pages 8697–8710, 2018. 1