Dynamic Channel and Layer Gating in Convolutional Neural Networks

Ali Ehteshami Bejnordi and Ralf Krestel

University of Passau
ehteshami.bahador@gmail.com, ralf.krestel@uni-passau.de

Abstract. Convolutional neural networks (CNN) are getting more and more complex, needing enormous computing resources and energy. In this paper, we propose methods for conditional computation in the context of image classification that allows a CNN to dynamically use its channels and layers conditioned on the input. To this end, we combine lightweight gating modules that can make binary decisions without causing much computational overhead. We argue, that combining the recently proposed channel gating mechanism with layer gating can significantly reduce the computational cost of large CNNs. Using discrete optimization algorithms, the gating modules are made aware of the context in which they are used and decide whether a particular channel and/or a particular layer will be executed. This results in neural networks that adapt their own topology conditioned on the input image. Experiments using the CIFAR10 and MNIST datasets show how competitive results in image classification with respect to accuracy can be achieved while saving up to 50% computational resources.

Keywords: Conditional Computation, Channel and Layer Gating, CNN, ResNet, Image Classification

1 Introduction

Conditional computation is a new emerging field in deep learning [4, 3]. Conditional computation aims to dynamically allocate resources in a neural network conditionally on the data. Conditional computation can be implemented in different ways such as dynamic execution of different sub-networks inside the main network or different layers or filters. Such models could allow running of different computation graphs conditioned on the input. For example, images that are easier may need less layers/filters or even shallower sub-branches in the network for making a prediction, while more complex examples may warrant the use of more computational resources in the network.

The most obvious benefits of conditional computation is saving resources at inference time. This is because the network is able to dynamically use parts of its units conditioned on the input. While the original base network can have a very large number of parameters and Multiply-accumulate operations (MAC), due to the dynamic structure of the network during inference, the resulting model may use a much lower average number of parameters and MACs.

The other advantage of conditional computation can be related to the formation of mixture of experts [11] inside the network. By gating individual components inside a network, we can make the parts that are active more specialized for the specific input, while other elements (filters/layers or sub-networks) may be specialized to perform well on other types of inputs. The idea of mixture of expert neural networks has been previously explored. Early works trained independent expert models to do different tasks and then joined the models and used a gating unit that could select the right model for the given input. Conditional computation, offers a generalized way of forming mixture of experts inside a neural network in which there is potentially a single expert model per example.

Dynamic capacity networks (DCN) [1] picked-up this idea by using a high capacity and a low capacity sub-network. The low capacity sub-network analyzes the full image, while the high capacity sub-network only focuses on task-relevant regions identified by the low capacity part.

In this paper, we propose to combine channel and layer gating using lightweight gating modules that can make binary decisions (1 for execution and 0 otherwise), saving computational costs, while maintaining high performance.

2 Related Work

There are several works in recent literature on conditional computation that successfully use gating to learn conditional layers/features in their networks. In this section we review and discuss several approaches for implementing conditional computation for computer vision applications.

2.1 Conditional Computation in Neural Networks

Stochastic Times Smooth Neurons. Bengio et al. [4] introduced stochastic times smooth neurons as gating units for conditional computation that can turn off large chunks of the computation performed within a deep neural network. The proposed gating units can produce actual zero for certain irrelevant inputs and hence lead to a sparsity that greatly reduces the computational cost of large deep networks. Even though stochastic gates perform non-smooth functions such as thresholds, the authors show that it is possible to obtain approximate gradients by introducing perturbations in the system and observing the effects.

The proposed stochastic neurons were used in the context of conditional computation to dynamically select parts of some computational graph for execution, given the current input. This work was among the first to show that a dynamic computational saving could be obtained without any significant loss in performance.

2.2 Layer Gating

Independence of Layers in Residual Neural Networks. Many of the modern CNNs are based on the recently proposed residual neural networks. In traditional architectures such as AlexNet [14] or VggNet [16], inputs are processed via low-level

features in the first few layers up to task specific high-level features in the very deep layers. However, the identity skip-connection in residual networks allows the data to flow from any layers to any subsequent layer [19]. In a study by Veit et al.[19], it was shown that removing single layers from residual networks at test time does not cause a significant drop in performance. This is in sharp contrast to traditional architectures such as AlexNet and VGG which have a dramatic performance drop after a layer removal. This shows that layers in ResNets[8] exhibit a significant degree of independence and that residual networks can be viewed as a collection of many paths, instead of a single very deep network. Motivated by these results, several methods were proposed for skipping execution of layers inside a network conditioned on the input such as convolutional neural networks with adaptive inference graph (ConvNet-AIG) [18] and SkipNet [20].

Convolutional Networks with an Adaptive Inference Graphs. Veit et al. [18] proposed a CNN architecture called convolutional neural networks with adaptive inference graph (ConvNet-AIG) that dynamically decides whether the current layer should be activated or not based on the input it receives. This allows constructing adaptive inference graphs conditionally on the input. This is achieved by training a set of gating units. Specifically, ConvNet-AIG [18] works with residual networks (ResNets) architecture [8] that are gated at each layer. The gating function is actually a basic neural network that can get the same featuremap that goes to a ResNet block as the input. The gating network makes a binary decision whether a layer should be enabled or turned off for the given input.

In ConvNet-AIG [18], the gating unit computes the global average pooling of the input and shrinks the entire featuremap into a vector size of $1 \times 1 \times C$, where C is the number of input channels. This vector is then passed to two fully connected layers that generate an output. This output has two nodes for two decisions: on meaning executing the layer, and off meaning skipping the layer. Technically, selecting the maximum between these two decisions for learning the gating function is a bad idea. If the network only considers this maximum decision, it might end up learning trivial solutions (For example, the gate may learn to remain always on or always off regardless of the input). Besides that taking the hard argmax function of the output is not differentiable. To circumvent this problem, the authors used the Gumbel-max trick [7]. Gumbel sampling is a strategy that allows us to sample from a discrete distribution.

The major limitation of this method is that it is not able to save computation on a more fine-grained level. In ConvNet-AIG[18], gates are only defined for each individual ResNet block (skip a whole block). It makes sense to enable more fine-grained gating such as gating of filters of the convolutional layers.

Skipping Layers Using Long Short-Term Memory Gates. SkipNet [20] is another method that can dynamically skip a layer in a ResNet architecture in a similar fashion. The authors use Long Short-term Memory (LSTM) [9] modules as the gating units for their network. For skipping redundant layers, the gating blocks of SkipNet[20] use a binary decision similar to ConvNet-AIG[18]. To overcome the problem of non-differentiable discrete decision Wang et al. [20] proposed a

hybrid algorithm that is a combination of supervised learning and reinforcement learning.

The gating module of SkipNet[20] is composed of a global average pooling layer and one 1×1 convolutional layer and also one LSTM layer. This recurrent gating design allows it to take benefits of LSTM architecture and reduces the cost of a CNN network inference time while at the same time achieving better results. SkipNet[20] bypasses fewer layers for difficult samples like dark or noisy images and skips more layers for easy images. However, in comparison to ConvNet-AIG[18] it achieves lower performance. One major problem with SkipNet[20] is that its accuracy drops rapidly as it saves more compute.

2.3 Gating Individual Filters/Channels

GaterNet [5] is a gating architecture that learns a complete convolutional network jointly and in parallel with the original network, and tries to gate individual filters in the base model. This method, however, comes at a high extra computation cost which may not be necessary. The authors proposed a gater network that extracts the features of the input and then based on these features gates the filters of main network. Chen et al. employed Improved SemHash trick[12] to make discrete gate functions differentiable during the backward pass. Unfortunately, the authors do not report any information about the MAC count or trade-off points between the accuracy and MAC-saving for different sparsity levels. This makes comparison of their approach with other methods challenging.

Dynamic channel pruning [6] is another recently proposed method that selects individual features to be turned on/off based on the input. This is done by choosing the top-k ranked features that should be executed for the specific input. Gao et al. [6] proposed using Feature Boosting and Suppression (FBS) method. This method uses auxiliary blocks that determine the importance of the output of a convolutional layer based on the input it receives. The authors showed that the Feature Boosting and Suppression method can improve the execution time 5 times faster than VGG-16 and 2 times faster than ResNet-18 while at the same time the reduction in accuracy is less than 0.6%.

Bejnordi proposed a model [2] which performs a more fine-grained level skipping by learning to execute filters inside a residual block conditioned on the input. The authors also proposed the batch-shaping loss to encourage the network to learn more conditional features. More recently, a new residual block was proposed where a gating module learns which spatial positions to evaluate exploiting the fact that not all regions in the image are equally important for the given task. The major benefit of our method is the ability of saving more computation cost rather than other methods. This architecture allows the model to gate a channel in a convolutional layer or a whole layer of residual block based on the input it receives.

3 Dynamic Layer and Channel Gating

In this work, we design a neural network architecture that enables fine-grained filter gating as well as layer-gating of whole residual blocks of convolutional neural networks. Unlike GaterNet [5] that gates individual channels using a learned auxiliary network, we use very light-weight gating modules similar to the ones used in ConvNet-AIG[18] for gating the filters and layers in each layer. Our proposed solution is called dynamic layer and channel gating (DLCG).

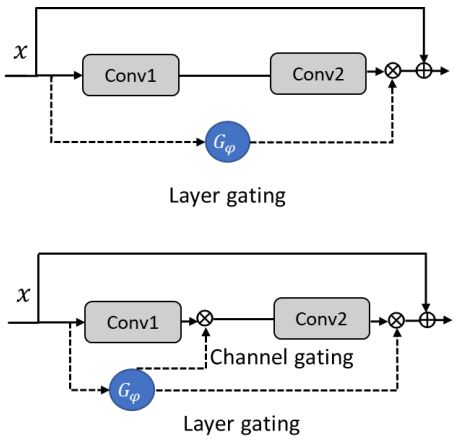

Fig. 1. Overview of different mechanism for gating a residual network. The top shows the layer gating approach proposed by Veit et al. [18]. The ResNet block in the bottom shows our proposed joint channel and layer gating (DLCG). In DLCG, we use a single gating module G_ϕ to jointly gate filters and layers

3.1 Proposed Gating Architecture

Figure 1 shows an overview of different gating strategies we consider in our work: Layer gating as proposed by Veit et al. [18], and our proposed joint channel and layer gating (DLCG). As shown in the lower part of the figure, we learn a single gating module to jointly gate layers and filters in the residual block.

3.2 The Structure of our Gating Module

The aim of the gating module G_ϕ is to estimate the relevance of a layer or filter given the input features. The gating module should have a light design (low MAC consumption) to not undermine the value of conditional computation, while at the same time operate in an input dependent fashion and make smart decisions for activating channels or layers in the block. Beside that, the gates should make

binary decisions. A gate with a soft output will not be useful. While a hard zero means we can skip the computation of a unit, a soft value such as 0.3 means we should still give some attention to the current unit and hence no computational saving will be obtained.

Our gating modules have a light and efficient structure inspired by the gating modules of ConvNet-AIG[18]. An overview of our gating module is presented in Figure 2. Our gating module takes the incoming featuremap to the residual block as input and applies a global average pooling to reduce the input dimension to $1 \times 1 \times N$, where N is the number of channels in the input featuremap. This step significantly reduces the computation costs of the gating network and is similarly used in Squeeze and Excitation networks [10]. This representation is obtained through:

$$z_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{i,j,c} \tag{1}$$

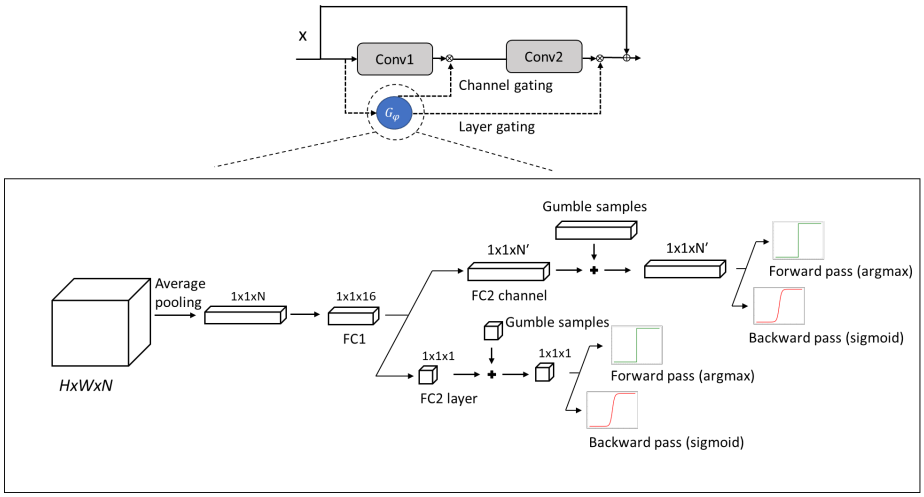

Fig. 2. Illustration of the gating module structure for our joint channel and layer gating network

This representation is then fed to a small MLP with a first fully connected layer with 16 neurons. This fully connected layer is a shared layer for both channel and layer gating. The output of this layer is passed to a ReLU non-linearity. After that, there are two separate heads, one for channel gating and one for layer gating (see Figure 2). The fully connected layer on the channel gating head generates the output probabilities for gating individual filters $\hat{\alpha}_i$, where $i \in \{1, 2, \cdots, N'\}$, and N' denotes the number of channels in the first convolutional layer of the ResNet block. Note that each element in the output

of this gating network is an independent binary gate $\hat{\alpha}_i$ responsible to choose to either execute or skip the computation of the filter i in the first convolutional layer of the ResNet block.

For layer gating, we use a fully connected layer that linearly projects the feature to a single output $\hat{\beta}$ (single gate) whose output determines if the current residual block should be executed or not. All the gates are trained using the Gumbel max trick[7] with sigmoid relaxation.

Note that during inference, we first look at the output of the layer gating and if it chooses to skip the layer, we do not perform any channel gating and the whole ResNet block is skipped. And in case the gate decides to execute the block, we proceed with the channel gating unit.

3.3 Sparsity Objective

Consider a gated classification model which only uses the task loss (e.g. categorical cross-entropy) to optimize the network. The gradients coming from the task loss could be back-propagated through the gating units. The most trivial solution for the gating units would be to make sure all the gates are always on. In this case, we end up with a network that is equivalent to a model trained without any gating units. Ideally, however, we would like the units and layers in the network to be input dependent. That means we want the gates to be on when the specific layer/filter is relevant for the current input and to be off if otherwise. To encourage this behaviour we use a sparsity objective which penalizes the gates for being always on.

Target Loss. In ConvNet-AIG[18], the sparsity is achieved by defining a loss function that encourages each layer to be executed at a certain target rate. The target rate can take a value between 0 and 1 representing the overall execution percentage of a layer. The execution rate is penalized in a mini-batch of data. The loss term is expressed as:

$$L_{target} = \sum_{l=1}^{N} (\bar{z}_l - t)^2 \tag{2}$$

in which t is the target rate and is a parameter selected by the user during training and $\bar{z}_l$ represents the fraction of images that are executed for a certain layer l and N is the total number of ResNet blocks. The total loss for optimizing the layer gated network is then obtained by summing up the normal loss function L_C (categorical cross-entropy) and the target rate loss L_{target}:

$$L_{AIG} = L_C + L_{target} \tag{3}$$

In practice, the best results in ConvNet-AIG[18] were achieved by manual setting of target rates per layer and following a lot of heuristics and hyper-parameter tuning. For example, the target rate of the initial layers and layers at the end of the network were set to 1 while the intermediate layers were given lower target rates as they seemed to be more prunable.

Target-Free Sparsity Loss. Unlike ConvNet-AIG[18], we propose to remove the target rate. This would allow different layers/channels to take varying dynamic execution rates. This way, the network may automatically learn to use more units for a specific layer and less for another, without us having to determine a target rate in advance. Besides that, we give weight to the sparsity loss by the coefficients λ and γ which control the pressure on the sparsity loss for layer gating and channel gating, respectively. The resulting loss equation for layer gating is, therefore:

$$L_{l-sparsity} = \sum_{l=1}^{N} \bar{z}_l^2 \tag{4}$$

And for the case of channel gating we have:

$$L_{ch-sparsity} = \sum_{f=1}^{K} \bar{z}_{ch}^2 \tag{5}$$

where z_{ch} denotes the fraction of images which activate a specific gate that gates whole layers, and K is the total number of filters that are gated in the network. Therefore, the final objective for joint channel and layer gating of our DLCG network is:

$$L_{DLCG} = L_C + \lambda L_{l-sparsity} + \gamma L_{ch-sparsity} \tag{6}$$

We optimize this loss with mini-batch stochastic gradient descent. To generate different sparsity levels for our gating network we set different values for our λ and γ coefficients.

4 Experiments

Evaluation Metrics. Top-1 and top-5 accuracies [14] are the measures that are used to evaluate the performance of algorithms for image classification tasks such as the ImageNet [14] or CIFAR [13] classification. Top-1 accuracy describes that the classifier gives the highest probability to the target label. Top-1 accuracy is also known as the normal accuracy and is widely used in benchmarks to rank different algorithms. Top-5 accuracy is mostly common when the number of classes are very large such as for ImageNet classification (1000 classes). Since we apply our model to MNIST and CIFAR10 classification tasks, we only report the top-1 accuracy.

Also for evaluating the computation cost of the model we report multiply-accumulate operations count (MAC). This measure gives us a good criterion of how fast our model is in practice. The computation time of a layer in a CNN architecture mostly depends on the MAC operations performed on that layer during the convolutional operation. The MAC count for a standard convolutional layer is $(H \times W \times C) \times (K \times K \times C')$. Where $(H \times W \times C)$ represents the dimension of the input featuremap and $(K \times K \times C')$ represents the spatial size of filters

Table 1. Results of the experiment on the CIFAR10 dataset for our joint channel and layer gating architecture (DLCG) with different sparsity loss coefficients

Gate loss factor		Average activation rate		Accuracy	GMAC
Layer	Channel	Layer	Channel		
0.09	0.15	0.399	0.215	88.96	0.0076
0.07	0.12	0.464	0.262	89.67	0.0089
0.09	0.10	0.405	0.272	89.87	0.0091
0.05	0.10	0.564	0.305	90.59	0.0115
0.05	0.07	0.677	0.388	91.55	0.0145
0.02	0.03	0.727	0.548	91.99	0.0206
0.00	0.03	0.992	0.591	92.49	0.0232
0.00	0.05	0.999	0.791	92.74	0.0320
0.00	0.00	0.999	0.880	92.86	0.0353

times the number of filters in the convolutional layer. By gating a specific filter we affect the number of filters C' that are applied to the input. Note that gating filters not only reduces the MAC count of the current layer, but also affects the MAC count of the following layer because the input dimension to the next layer is automatically reduced.

To get better insight into the effectiveness of the gating architectures, we also plot the MAC versus Accuracy curve to see how saving computation affects the accuracy of the gated models.

4.1 Experiments on CIFAR10

For evaluation of our approach, we use the CIFAR10 dataset [13]. CIFAR10 is a popular dataset for the task of image classification consisting of 10 categories It contains of 50000 images for training and 10000 images for testing of size 32×32 pixels.

Training Configuration for CIFAR10 Classification. We used ResNet20 [8] as the base network for our CIFAR10 experiments. We trained our joint layer and channel gated models using stochastic gradient descent with Nesterov momentum [17]. The network was trained for a total of 400 epochs with a batch size of 256. At the start of training, the learning rate was set to 0.1. We followed a step policy for learning rate drop and divided the initial learning rate by a factor of 10 at epochs 200, 300 and 375. The weight decay for the parameters of the network was set to $5e^{-4}$. We did not apply weight decay to any of the parameters of the gating modules (weights or biases).

We used random cropping and random horizontal flipping as data augmentation to improve the generalization of our model. To generate trade-off points for our MAC-accuracy curve, we experimented with different values of λ and γ for the sparsity objectives.

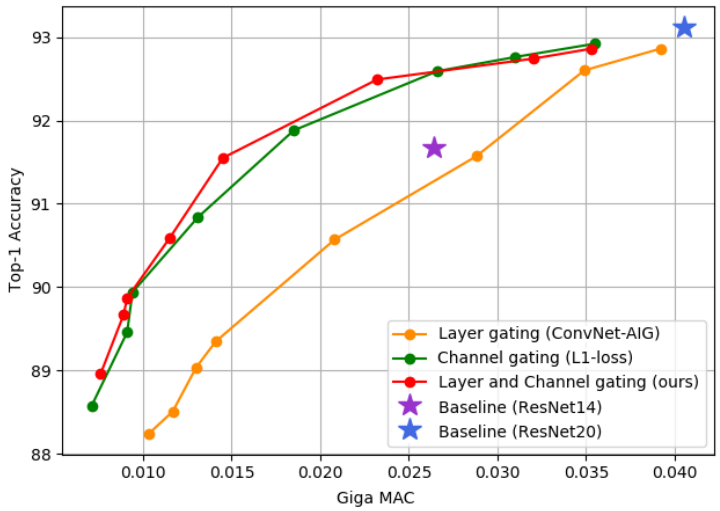

Fig. 3. The top-1 accuracy vs MAC count curve for the three gating architectures trained on CIFAR10 dataset

Results on CIFAR10: We present the result of our DLCG model in Table 1 with different accuracy vs mac trade-offs. This model is performing consistently better than ConvNet-AIG[18] as shown in Figure 3. We additionally compare the results to the case in which we only use channel gating. The accuracy of our DLCG model is slightly higher than a sole channel gated model as well. We argue that the major performance gain comes from the channel gating modules and that is clear from the significant performance gap between channel gating alone and ConvNet-AIG[18].

In Figure 3, we show the trade-off between MAC count and accuracy for the three different gating schemes: ConvNet-AIG[18] and channel gating as well as joint channel and layer gating. Note that in all MAC count calculations, we also include the overhead of the gating modules (less than 0.03%).

From the results, it is obvious that our proposed gating models outperform ConvNet-AIG[18] by a large margin. In this plot we also present the performance of two baseline models: ResNet20 and ResNet14 without any gating. As can be seen, ResNet14 without gating outperforms a gated ConvNet-AIG[18] model at a similar MAC count. This result is surprising, because in such a case one would prefer to use a ResNet14 model rather than a ConvNet-AIG[18] model with a ResNet20 backbone. This result questions the entire value of conditional computation. Our ResNet20 based gated models, in contrast, outperform ResNet14 non-gated baseline by a large margin at a similar MAC count. This is highly desirable, as it means we can take a large capacity neural network (such as ResNet20) and sparsify it to the size of a smaller network (such as ResNet14), while getting a much higher accuracy than the smaller non-gated model.

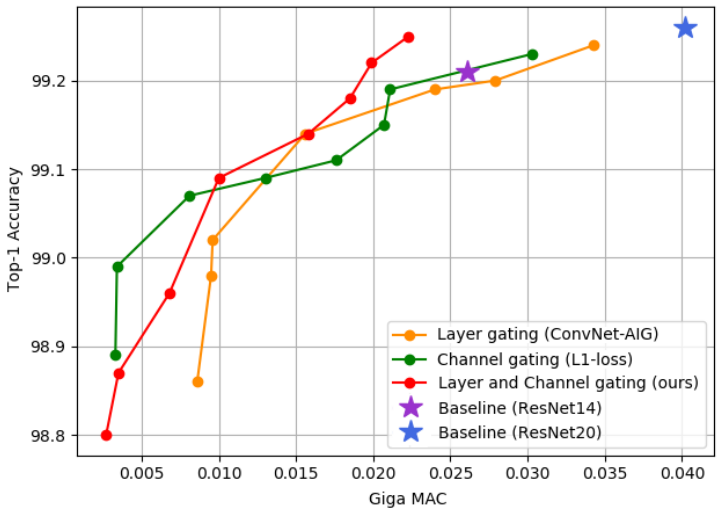

Fig. 4. The top-1 accuracy vs MAC count curve for the three gating architectures trained on MNIST dataset

4.2 Experiments on MNIST

To verify our results from the CIFAR10 dataset, we additionally evaluated our approach on the smaller MNIST dataset [15]. MNIST is a database of handwritten digits from 0 to 9. Each image is available in the form of a grayscale image with a size of 28×28 pixels. This dataset contains 60000 images for training and 10000 images for testing.

Figure 4 shows the top-1 accuracy versus MAC count curve for the three gating scenarios: layer gating, channel gating, and joint channel and layer gating (DLCG). As can be seen, our DLCG model outperforms the channel gating model and also the ConvNet-AIG[18] model (layer gating) in high accuracy ranges. DLCG outperforms ConvNet-AIG[18] at all trade-off points and shows that the addition of a more fine-grained gating mechanism could be beneficial for conditional computation neural networks.

5 Conclusions and Future Work

In this paper, we studied conditional computation models for vision applications. An important limitation in conventional neural network architectures is their fixed static graph. The deep learning models we train for various tasks are largely task- and context-agnostic. This implies that regardless of the input, all elements of the network are executed. This shortcoming may render such models inefficient in many real-world applications such as running models on mobile devices. Therefore, we focused on the design of a convolutional neural network that

can dynamically utilize its units conditioned on the input image. In particular, we presented a joint layer and channel gating architecture, that can decide to activate or deactivate channels in a convolutional layer or a whole residual block based on their relevance to the specific input.

Our empirical evaluations show that channel gating alone can outperform layer gating methods such as ConvNet-AIG [18] by a large margin on the MNIST and CIFAR10 datasets. This increase in performance could be attributed to the fine-grained nature of our architecture design. Rather than saying a whole residual block with all its computation units are irrelevant for the input, we decide the computation saving at the fine-grained channel level. Our joint layer and channel gating show some improvement over channel gating alone, but not significantly as most of the computational saving comes from the channel gating operations. Overall, our proposed gating architecture provides improved efficiency and classification accuracy.

We speculate that the reason why channel gating alone may perform as good as joint channel and layer gating could be as follows. The gating module generally produces very sparse channel gating solutions in each ResNet block which would lead to significant saving in computation. However, the small number of filters which are remaining active are necessary to achieve a high performance. Therefore, the model generally prefers to choose some filters from each layer (albeit small in number) rather that skipping the entire block.

A highly desirable aspect of our proposed gating approach is that we can take a large capacity neural network such as ResNet20 and sparsify it to the size of a smaller network such as ResNet14, while achieving a much higher accuracy than this small non-gated model (ResNet14).

There are many future research directions. One would be to use the batch-shaping loss proposed in [2] for our model. We think our joint channel and layer gating architecture could potentially benefit from this loss. It would additionally be useful to evaluate the performance of our method on the larger scale ImageNet dataset.

Another direction would be to integrate early exiting methods to our model to not only save computation by gating individual filters and layers but also exit the whole model at an early stage in case the model is already certain about the decision regarding an easy example. This way, the model can choose to skip huge amount of computation for easier examples.

References

1. Almahairi, A., Ballas, N., Cooijmans, T., Zheng, Y., Larochelle, H., Courville, A.: Dynamic capacity networks. In: International Conference on Machine Learning. pp. 2549–2558 (2016)
2. Bejnordi, B.E., Blankevoort, T., Welling, M.: Batch-shaping for learning conditional channel gated networks. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=Bke89JBtvB
3. Bengio, E., Bacon, P.L., Pineau, J., Precup, D.: Conditional computation in neural networks for faster models. arXiv preprint arXiv:1511.06297 (2015)

4. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
5. Chen, Z., Li, Y., Bengio, S., Si, S.: You look twice: Gaternet for dynamic filter selection in cnns. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
6. Gao, X., Zhao, Y., Łukasz Dudziak, Mullins, R., zhong Xu, C.: Dynamic channel pruning: Feature boosting and suppression. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=BJxh2j0qYm
7. Gumbel, E.J.: Statistical theory of extreme values and some practical applications. NBS Applied Mathematics Series **33** (1954)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
10. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
11. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E., et al.: Adaptive mixtures of local experts. Neural computation **3**(1), 79–87 (1991)
12. Kaiser, Ł., Bengio, S.: Discrete autoencoders for sequence models. arXiv preprint arXiv:1801.09797 (2018)
13. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
15. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
16. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
17. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International conference on machine learning. pp. 1139–1147 (2013)
18. Veit, A., Belongie, S.: Convolutional networks with adaptive inference graphs. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 3–18 (2018)
19. Veit, A., Wilber, M., Belongie, S.: Residual networks behave like ensembles of relatively shallow networks. Conference on Neural Information Processing Systems (NIPS) (2016)
20. Wang, X., Yu, F., Dou, Z.Y., Darrell, T., Gonzalez, J.E.: Skipnet: Learning dynamic routing in convolutional networks. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 409–424 (2018)