Plug-in, Trainable Gate for Streamlining Arbitrary Neural Networks

Jaedeok Kim,[1*] **Chiyoun Park,**[1*] **Hyun-Joo Jung,**[1*] **Yoonsuck Choe**[1,2]
[1]Artificial Intelligence Center, Samsung Research, Samsung Electronics Co.
56 Seongchon-gil, Secho-gu, Seoul, Korea, 06765
[2]Department of Computer Science and Engineering, Texas A&M University
College Station, TX, 77843, USA
{jd05.kim, chiyoun.park, hj34.jung}@samsung.com, choe@tamu.edu

Abstract

Architecture optimization, which is a technique for finding an efficient neural network that meets certain requirements, generally reduces to a set of multiple-choice selection problems among alternative sub-structures or parameters. The discrete nature of the selection problem, however, makes this optimization difficult. To tackle this problem we introduce a novel concept of a trainable gate function. The trainable gate function, which confers a differentiable property to discrete-valued variables, allows us to directly optimize loss functions that include non-differentiable discrete values such as 0-1 selection. The proposed trainable gate can be applied to pruning. Pruning can be carried out simply by appending the proposed trainable gate functions to each intermediate output tensor followed by fine-tuning the overall model, using any gradient-based training methods. So the proposed method can jointly optimize the selection of the pruned channels while fine-tuning the weights of the pruned model at the same time. Our experimental results demonstrate that the proposed method efficiently optimizes arbitrary neural networks in various tasks such as image classification, style transfer, optical flow estimation, and neural machine translation.

Introduction

Deep neural networks have been widely used in many applications such as image classification, image generation, and machine translation. However, in order to increase accuracy of the models, the neural networks have to be made larger and require a huge amount of computation (He et al. 2016a; Simonyan and Zisserman 2014). Because it is often not feasible to load and execute such a large model on an on-device platform such as mobile phones or IoT devices, various architecture optimization methods have been proposed for finding an efficient neural network that meets certain design requirements. In particular, pruning methods can reduce both model size and computational costs effectively, but the discrete nature of the binary selection problems makes such methods difficult and inefficient (He et al. 2018; Luo and Wu 2018).

Gradient descent methods can solve a continuous optimization problem efficiently by minimizing the loss function, but such methods are not directly applicable to discrete optimization problems because they are not differentiable. While many alternative solutions such as simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) have been proposed to handle discrete optimization problems, they are too cost-inefficient in deep learning because we need to train alternative choices to evaluate the sample's accuracy.

In this paper, we introduce a novel concept of a trainable gate function (TGF) that confers a differentiable property to discrete-valued variables. It allows us to directly optimize, through gradient descent, loss functions that include discrete choices that are non-differentiable. By applying TGFs each of which connects a continuous latent parameter to a discrete choice, a discrete optimization problem can be relaxed to a continuous optimization problem.

Pruning a neural network is a problem that decides which channels (or weights) are to be retained. In order to obtain an optimal pruning result for an individual model, one needs to compare the performance of the model induced by all combinations of retained channels. While specialized structures or searching algorithms have been proposed for pruning, they have complex structures or their internal parameters need to be set manually (He et al. 2018). The key problem of channel pruning is that there are discrete choices in the combination of channels, which makes the problem of channel selections non-differentiable. Using the proposed TGF allows us to reformulate discrete choices as a simple differentiable learning problem, so that a general gradient descent procedure can be applied, end-to-end.

Our main contributions in this paper are in three fold.

- We introduce the concept of a TGF which makes a discrete selection problem solvable by a conventional gradient-based learning procedure.
- We propose a pruning method with which a neural network can be directly optimized in terms of the number of parameters or that of FLOPs. The proposed method can prune and train a neural network simultaneously, so that the further fine-tuning step is not needed.
- Our proposed method is task-agnostic so that it can be easily applied to many different tasks.

[*] Equal contribution.

Simply appending TGFs, we have achieved competitive results in compressing neural networks with minimal degradation in accuracy. For instance, our proposed method compresses ResNet-56 (He et al. 2016a) on CIFAR-10 dataset (Krizhevsky and Hinton 2009) by half in terms of the number of FLOPs with negligible accuracy drop. In a style transfer task, we achieved an extremely compressed network which is more than 35 times smaller and 3 times faster than the original network. Moreover, our pruning method has been effectively applied to other practical tasks such as optical flow estimation and neural machine translation.

By connecting discrete and continuous domains through the concept of TGF, we are able to obtain competitive results on various applications in a simple way. Not just a continuous relaxation, it directly connects the deterministic decision to a continuous and differentiable domain. By doing so, the proposed method in this paper could help us solve more practical applications that have difficulties due to discrete components in the architecture.

Related Work

Architecture optimization can be considered as a combinatorial optimization problem. The most important factors are to determine which channels should be pruned within a layer in order to minimize the loss of acquired knowledge.

These can be addressed as a problem that finds the best combination of retained channels where it requires extremely heavy computation. As an alternative, heuristic approaches have been proposed to select channels to be pruned (He, Zhang, and Sun 2017; Li et al. 2017). Although these approaches provide rich intuition about neural networks and can be easily adopted to compress a neural network quickly, such methods tend to be sub-optimal for a given task in practice.

The problem of finding the best combination can be formulated as a reinforcement learning (RL) problem and then be solved by learning a policy network. Bello et al. (Bello et al. 2017) proposed a method to solve combinatorial optimization problems including traveling salesman and knapsack problems by training a policy network. Zoph and Le (Zoph and Le 2016) proposed an RL based method to find the most suitable architecture. The same approach can be applied to find the best set of compression ratio for each layer that satisfies the overall compression and performance targets, as proposed in (He et al. 2018; Zhong et al. 2018). However, RL based methods still require extremely heavy computation.

To tackle the scalability issue, a differentiable approach has been considered in various research based on continuous relaxation (Liu, Simonyan, and Yang 2019; Liu et al. 2017; Louizos, Ullrich, and Welling 2017). To relax a discrete problem to be differentiable, Liu et al. (Liu, Simonyan, and Yang 2019) proposed a method that places a mixture of candidate operations by using softmax. Luo and Wu (Luo and Wu 2018) proposed a type of a self-attention module with a scaled sigmoid function as an activation function to retain channels from probabilistic decision. However, in these methods it is essential to carefully initialize and control the parameters of the attention layers and the scaled sigmoid function. While a differentiable approach is scalable to a large search space, existing approaches determine the set of selected channels in a probabilistic way so that they require an additional step to decide whether to prune each channel or not.

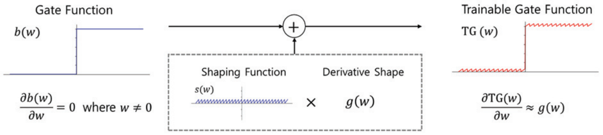

Gate Function
$b(w)$
$\frac{\partial b(w)}{\partial w} = 0$ where $w \neq 0$
Shaping Function
Derivative Shape
$s(w)$
×
$g(w)$
Trainable Gate Function
TG (w)
$\frac{\partial \mathrm{TG}(w)}{\partial w} \approx g(w)$

Figure 1: Explanation of how to shape a gate function to be trainable. The original gate function has zero derivative as in the left. We add to the original gate function the shaping function $s(w)$ multiplied by a desirable derivative shape $g(w)$ which changes the derivative of the gate function. The resulting function is a TGF that has a desirable derivative.

The method we propose here allows us to find the set of channels deterministically by directly optimizing the objective function which confers a differentiable property to discrete-valued variables, thus bypassing the additional step required in probabilistic approaches. The proposed optimization can be performed simply by appending TGFs to a target layer and train it using a gradient descent optimization. The proposed method does not depend on additional parameters, so that it does not need a careful initialization or specialized annealing process for stabilization.

Differentially Trainable Gate Function

Consider combinatorial optimization in a selection problem.

$$\min_{\theta \in \Theta, \mathbf{b} \in \{0,1\}^n} \mathcal{L}(\theta, \mathbf{b}) \tag{1}$$

where $\mathbf{b} = (b_1, \cdots, b_n)$ is a vector of binary selections. $\mathcal{L}\colon \Theta \times \{0,1\}^n \to \mathbb{R}$ is an objective function parameterized by θ. The optimization problem (1) is a generalized form of a selection problem that can cover a parameterized loss function such as a neural network pruning problem. In case of a pure selection problem we set the domain Θ to be a singleton set.

To make the problem differentiable, we consider b_i as an output of a binary gate function $b\colon \mathbb{R} \to \{0,1\}$ parameterized by an auxiliary variable w_i. We will let b be a step function for convenience. [1] Then the optimization problem (1) is equivalent to the following.

$$\min_{\theta \in \Theta, \mathbf{w} \in \mathbb{R}^n} \mathcal{L}(\theta, b(w_1), \cdots, b(w_n)) \tag{2}$$

where the problem is defined in continuous domain. That is, if $(\theta^*, \mathbf{w}^*)$ is a global minimum of (2), then $(\theta^*, b(\mathbf{w}^*))$ is a global minimum of (1).

While the continuous relaxation (2) enables the optimization problem (1) to be solved by gradient descent, a gate

[1] Although we only consider a step function as a gate function b, the same argument can be easily applied to an almost everywhere differentiable binary gate function.

function $b_i(\cdot)$ has derivative of zero wherever differentiable and consequently

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial b}\frac{\partial b}{\partial w_i} = 0, \text{ where } w_i \neq 0. \tag{3}$$

So, a gradient descent optimization does not work for such a function.

In order to resolve this issue, we consider a new type of gate function which has non-zero gradient and is differentiable almost everywhere. Motivated by (Hahn and Choi 2018), we first define a gradient shaping function $s\colon \mathbb{R} \to \mathbb{R}$ by

$$s^{(M)}(w) := \frac{Mw - \lfloor Mw \rfloor}{M} \tag{4}$$

where M is a large positive integer and $\lfloor w \rfloor$ is the greatest integer less than or equal to w. Note that this function has near-zero value for all w, and its derivative is always one wherever differentiable. Using (4) we consider a trainable gate defined as the following (see Figure 1).

Definition 1 *A function* $\mathrm{TG}^{(M)}\colon \mathbb{R} \to \mathbb{R}$ *is said to be a trainable gate of a gate function* $b\colon \mathbb{R} \to \{0, 1\}$ *with respect to a gradient shape* $g\colon \mathbb{R} \to \mathbb{R}$ *if*

$$\mathrm{TG}^{(M)}(w; g) := b(w) + s^{(M)}(w)g(w). \tag{5}$$

Then a trainable gate $\mathrm{TG}^{(M)}$ satisfies the following proposition.

Proposition 1 *For any bounded derivative shape* g *whose derivative is also bounded,* $\mathrm{TG}^{(M)}(w; g)$ *uniformly converges to* $b(w)$ *as* $M \to \infty$*. Moreover,* $\mathrm{TG}^{(M)'}(w; g)$ *uniformly converges to* $g(w)$*.*

Proof. By definition (5), it satisfies that for all $w \in \mathbb{R}$

$$\begin{aligned} |\mathrm{TG}^{(M)}(w; g) - b(w)| &= |s^{(M)}(w)g(w)| \\ &\leq \frac{1}{M}|g(w)| \to 0, \end{aligned}$$

as $M \to \infty$.

Also, $s'(w) = 1$ if $s(w)$ is differentiable at w, which yields for all $w \in \mathbb{R}$

$$\begin{aligned} &|\mathrm{TG}^{(M)}(w; g)'(w) - g(w)| \\ &= |s^{(M)'}(w)g(w) + s^{(M)}(w)g'(w) - g(w)| \\ &= |s^{(M)}(w)g'(w)| \\ &\leq \frac{1}{M}|g'(w)| \to 0, \end{aligned}$$

as $M \to \infty$. ■

Proposition 1 guarantees that the trainable gate $\mathrm{TG}^{(M)}(w; g)$ can approximate the original gate function $b(w)$, while its derivative still approximates the desired derivative shape g. It is now possible to control the derivative of the given kernel $b(w)$ as we want and hence a gradient descent optimization is applicable to the TGF $\mathrm{TG}^{(M)}(w; g)$. For convenience, we will drop the superscript and the desired function g from $\mathrm{TG}^{(M)}(w; g)$ unless there is an ambiguity.

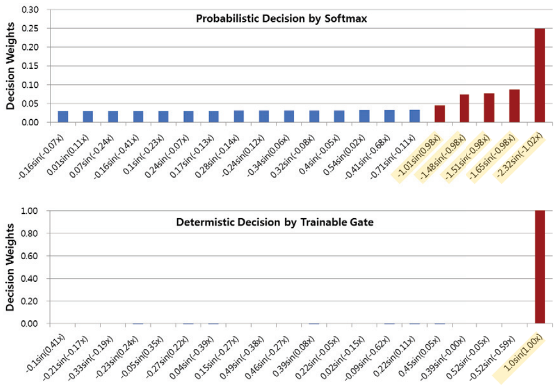


Figure 2: Difference between probabilistic and deterministic decisions. Each bar in the above graph indicates the decision weight of a hidden node learnt by a probabilistic method. The ith label on the x-axis represents the learnt weights w_i and c_i of the ith hidden node. The graph below shows the results from a TGF where there is no redundant hidden node.

Difference between Probabilistic and Deterministic Decisions

The proposed TGF directly learns a deterministic decision during the training phase unlike existing methods. While a probabilistic decision has been considered in existing differentiable methods (Jang, Gu, and Poole 2017; Liu, Simonyan, and Yang 2019; Liu et al. 2017; Louizos, Ullrich, and Welling 2017), probabilistic decisions are not clear to select and hence it needs further decision steps. Due to the on-off nature of our TGF's decision, we can include more decisive objectives, such as the number of channels, FLOPs or parameters, without requiring approximation to expectation of the distribution or smoothing to non-discrete values.

We performed a synthetic experiment in order to see the difference between probabilistic and deterministic approaches. To this end, we generate a training dataset to learn the sine function $\sin(\cdot)$. Consider a neural network having a single fully-connected layer having 20 hidden nodes each of which adopts the sine function as its activation. Since a training sample in our synthetic dataset is of the form $(x, \sin(x))$, it is enough to use only one hidden node to express the relation between input x and output y.

We consider a selection layer consisting of a continuous relaxation function with 20 hidden nodes each of which learns whether to retain the corresponding hidden node or not. Two different types of a relaxation function are addressed: a softmax function for a probabilistic decision (Liu, Simonyan, and Yang 2019) and the proposed TGFs for a deterministic decision.

As we can see in Figure 2, the probabilistic method found a solution that uses more than one node. In particular, the top 5 hidden nodes based on the decision weight have similar weight values w_i and c_i. In a training phase the probabilistic decision uses a linear combination of options so that error can be canceled out and as a result the selections will be re-

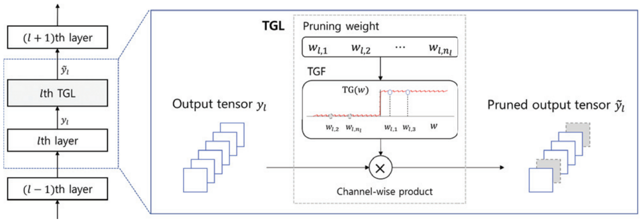


Figure 3: The overview of the proposed TGL. The figure shows an example of a TGL appending to a convolution layer. The TGL consists of a set of gate functions each of which determines whether to prune or not. Each gate function outputs a value of 0 or 1 which is multiplied to the corresponding filter of the convolution kernel of the target layer.

dundant. On the other hand, the deterministic decision (our proposed TGF) selects only one node exactly. Due to the on-off nature of the deterministic decision, the TGF learns by incorporating the knowledge of selections. So the deterministic decision can choose effectively without redundancy.

While we have considered the binary selection problem so far, it is straightforward to extend the concept of a trainable gate to an n-ary case by using n-simplex. However, in order to show the practical usefulness of the proposed concept, we will consider the pruning problem, an important application of a TGF, in which it is enough to use a binary gate function.

Differentiable Pruning Method

In this section we develop a method to efficiently and automatically prune a neural network as an important application of the proposed TGF. To this end, using the concept of a TGF we propose a trainable gate layer (TGL) which learns how to prune channels from its previous layer.

Design of a Trainable Gate Layer

The overall framework of our proposed TGL is illustrated in Figure 3. Channel pruning can be formulated by a function that zeros out certain channels of an output tensor in a convolutional neural network and keeps the rest of the values. We thus design a TGL as a set of TGFs whose elements correspond to the output channels of a target layer. Let the lth target layer map an input tensor x_l to an output tensor $y_l := (y_{l,1}, \cdots, y_{l,n_l})$ that has n_l channels using a kernel K_l

$$y_{l,i} = K_{l,i} * x_l.$$

for $i = 1, \cdots, n_l$. A fully connected layer uses the multiplication × instead of a convolution operation $*$, but we here simply use $*$ to represent both cases.

Let a TGL P_l prune the lth target layer where the TGL P_l consists of trainable weights $\mathbf{w}_l := (w_{l,1}, \cdots, w_{l,n_l})$ and a function $\mathrm{TG}(\cdot)$. The weight $w_{l,i}$, $i = 1, \cdots, n_l$, is used to learn whether the corresponding channel should be masked by zero or not. The TGL P_l masks the output tensor y_l of the lth target layer as

$$\tilde{y}_{l,i} = \mathrm{TG}(w_{l,i}) y_{l,i}, \tag{6}$$

where $\tilde{y}_l := (\tilde{y}_{l,1}, \cdots, \tilde{y}_{l,n_l})$ is the pruned output tensor by P_l. Since we have $y_{l,i} = K_{l,i} * x_l$, (6) can be rewritten as

$$\begin{aligned}\mathrm{TG}(w_{l,i}) y_{l,i} &= \mathrm{TG}(w_{l,i})(K_{l,i} * x_l) \\ &= (\mathrm{TG}(w_{l,i}) K_{l,i}) * x_l.\end{aligned}$$

So multiplying $\mathrm{TG}(w_{l,i})$ to $y_{l,i}$ masks the ith channel from the kernel K_l. While $\mathrm{TG}(w_{l,i})$ might not be exactly zero due to the gradient shaping, its effect can be negligible by letting the value of M be large enough.

From (6), $\mathrm{TG}(w_{l,i}) = 0$ yields $\tilde{y}_{l,i} = 0$. So the value of the weight $w_{l,i}$ can control the ith channel of the output tensor y_l. If a step function $b(w) = \mathbf{1}_{[w>0]}$ is used as the gate function in the TGL, $w_{l,i} < 0$ implies that the TGL zeros out the ith channel from the lth layer. Otherwise, the channel remains identical. Hence, by updating the weights w_l, we can make the TGL learn the best combination of channels to prune.

The proposed channel pruning method can be extended to weight pruning or layer pruning in a straightforward way. It can be achieved by applying the trainable gate functions for each elements of the kernel or each layer.

Compression Ratio Control

The purpose of channel pruning is to reduce neural network size or computational cost. However, simply adding the above TGL without any regularization does not ensure that channels are pruned out as much as we need. Unless there is a significant amount of redundancy in the network, having more filters is often advantageous to obtain higher accuracy, so the layer will not be pruned. Taking this issue into account, we add a regularization factor to the loss function that controls the compression ratio of a neural network as desired.

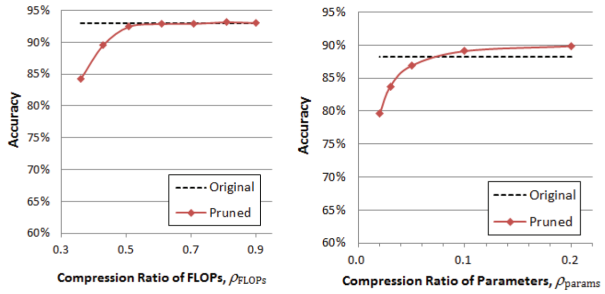
Let ρ be the target compression ratio. In case of reducing the number of FLOPs, the target compression ratio ρ is defined by the ratio between the number of the remaining FLOPs and the total number C_{tot} of FLOPs of the neural network. The weight values $\mathbf{w} := (\mathbf{w}_1, \cdots, \mathbf{w}_L)$ of TGLs determine the remaining number of FLOPs, denoted by $C(\mathbf{w})$. We want to reduce the FLOPs of the pruned model by the factor of ρ, that is, we want it to satisfy $C(\mathbf{w})/C_{tot} \approx \rho$.

Let $\mathcal{L}(\theta, \mathbf{w})$ be the original loss function to be minimized where θ denotes the weights of layers in the neural network except the TGLs. We add a regularization term to the loss function as follows in order to control the compression ratio.

$$\mathcal{L}(\theta, \mathbf{w}; \rho) := \mathcal{L}(\theta, \mathbf{w}) + \lambda \left\| \rho - C(\mathbf{w})/C_{tot} \right\|_2^2 \tag{7}$$

where $\|\cdot\|_2$ denotes the l_2-norm and λ is a regularization parameter. The added regularization will ensure that the number of channels will be reduced to meet our desired compression ratio.

Note that minimization of the loss function (7) does not only update the weights of TGLs but also those of the normal layers. A training procedure, therefore, jointly optimizes the selection of the pruned channels while fine-tuning the weights of the pruned model at the same time. In traditional approaches where channel pruning procedure is followed by a separate fine-tuning stage, the importance of each channel



(a) ResNet-56 (CIFAR-10)

(b) VGG-16 (ImageNet)

Figure 4: Pruning results without fine-tuning. (a) Channel pruning of ResNet-56 and (b) Weight pruning of VGG-16 The x-axis represents the compression ratio which is the ratio of the remaining of a pruned model over the original number of FLOPs or parameters.

may change during fine-tuning stage, which leads to sub-optimal compression. Our proposed method, however, does not fall into such a problem since each TGL automatically takes into account the importance of each channel while adjusting the weights of the original model based on the pruned channels. The loss function indicates that there is a trade-off between the compression ratio and accuracy.

While we have considered the number of FLOPs in this subsection, we can extend easily to other objectives such as the number of weight parameters or channels by replacing the regularization target in (7).

Experimental Results

In this section, we will demonstrate the effectiveness of the proposed TGF through various applications in the image and language domains. We implemented our experiments using Keras (Chollet and others 2015) unless mentioned otherwise. In order to shape the derivative of the gate function b, a constant derivative shape $g(w) = 1$ and $M = 10^5$ is used and a simple random weight initialization are used. In all experiments, only convolution or fully-connected layers are considered in calculating the number of FLOPs of a model since the other type of layers, e.g., batch normalization, requires relatively negligible amount of computation.

Image Classification

We used CIFAR-10 and ImageNet datasets for our image classification experiments. We used pretrained weights of ResNet-56 on CIFAR-10 that trained from scratch with usual data augmentations (normalization and random cropping). For VGG-16 on ImageNet we used pre-trained weights that was published in Keras. Although we found that the accuracy of each model in our experimental setup differs slightly from the reported value, the original pre-trained weights were used in our experiments without modification since we wanted to investigate the performance of our proposed method in terms of the accuracy drop.

Pruning without Fine-tuning In order to show the effect of the TGFs, we first considered a selection problem. We kept the pre-trained weights of a model and only pruned the channels or weights without fine-tuning by appending TGLs to convolution and fully-connected layers. That is, we fix the weights θ and optimize $\mathbf{w}$ in (2). Figure 4a shows the results of channel pruning in ResNet-56 (He et al. 2016b) on CIFAR-10. It can be observed that the number of FLOPs can be reduced by half without noticeable change in accuracy even when we do not apply fine-tuning, which implies that the TGFs works as expected.

Table 1: Channel pruning of ResNet-56 on the CIFAR-10 and weight pruning of VGG-16 on ImageNet. acc and Δ_{acc} denote the accuracy after pruning and the accuracy drop, respectively. ρ_{FLOPs} (ρ_{params}, resp.) means the number of remaining FLOPs (parameters, resp.) over the total number of FLOPs (parameters, resp.) in each model, where a small value means more pruned. Methods: FP (Li et al. 2017), CP (He, Zhang, and Sun 2017), VCP (Zhao et al. 2019), AMC (He et al. 2018), ADMM and PWP (Ye et al. 2018).

Model	Method	Δ_{acc} (acc) (%)	ρ
ResNet-56 ($\rho = \rho_{\text{FLOPs}}$)	FP	0.02 (93.06)	0.72
	VCP	-0.8 (92.26)	0.79
	Proposed	**0.04** (92.7)	0.71
	CP	-1.0 (91.8)	0.5
	AMC	-0.9 (91.9)	0.5
	Proposed	**-0.3** (92.36)	0.51
VGG-16 ($\rho = \rho_{\text{params}}$)	ADMM	0.0 (88.7)	1/19
	Proposed	**0.76** (89.0)	1/20
	PWP	-0.5 (88.2)	1/34
	Proposed	**-0.06** (88.18)	1/50

We also applied weight pruning mentioned in the previous section to VGG-16 on ImageNet (Figure 4b). Even though there are more than 1.38×10^8 weight parameters in the model whether to be retained or not, simply plugging-in TGLs to the model allows us to find a selection of redundant weight parameters. Note that the accuracy of VGG-16 even increases to 89.1% from 88.24% when only 10% of the parameters are retained. This phenomenon is due to the fact that reducing the number of non-gated parameters are equivalent to applying L_0 regularization to the neural network, and so adding the gating function to each parameter improves the generalization property of the model further.

Pruning with Fine-tuning In the next example, we jointly optimized the weights and selection at the same time in order to incorporate fine-tuning to the selections. Like the previous example we appended TGLs to a model, but we jointly trained both the TGLs and the weight parameters of the model. Table 1 summarizes the pruning results. As shown in the table, our results are competitive with existing methods. For example, in ResNet-56, the number of FLOPs is reduced by half while maintaining the original accuracy. It is also noticeable that we achieve higher accuracy on the compressed VGG-16 model, even if the accuracy of our initial model was worse.

While in our experiments we used a constant function to shape the derivative within the TGFs, The proposed method can adopt any derivative shape by changing $g(w)$ in (5). Fig-

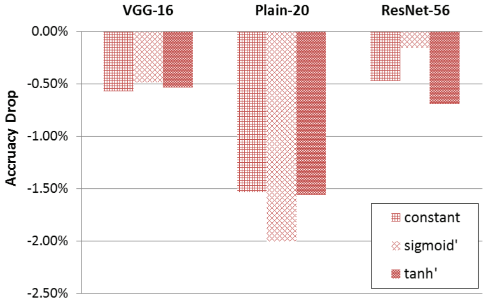


Figure 5: Comparing the effect of gradient shaping in TGFs. *sigmoid'* denotes the derivative of the sigmoid function $\sigma(w) = 1/(1+e^{-x})$ and *tanh'* denotes the derivative of the hyperbolic tangent function.

Table 2: A summary of style transfer model compression results. We measured the size of model files which are saved in binary format (.pb). The inference time was averaged over 10 frames on the CPU of a Galaxy S10. The frame size is 256×256 pixels.

Model	File size (MB)	FLOPs (G)	Params (M)	Time (ms)
Original	6.9	10.079	1.674	549
$\rho_{ch} = 0.4$	1.8	2.470	0.405	414
$\rho_{ch} = 0.1$	0.2	0.227	0.020	160

ure 5 compares the effect of different shaping functions. It shows that the derivative shape g does not affect the results critically, which implies that our proposed method is stable over a choice of $g(w)$. It can be concluded that a simple constant derivative shape $g(w) = 1$ can be adopted without significant loss of accuracy.

Image Generation

We further applied our proposed pruning method to style transfer and optical flow estimation models which are the most popular applications in image generation.

Style Transfer Style transfer (Dumoulin, Shlens, and Kudlur 2017; Gatys, Ecker, and Bethge 2015) generates a new image by synthesizing contents in given content image with styles in given style image. Because a style transfer network is heavily dependent on the selection of which styles are used, it is not easy to obtain proper pre-trained weights. So, we started from N-style transfer network (Dumoulin, Shlens, and Kudlur 2017) as an original architecture [2] with *randomly initialized weights*. It is of course possible to start from pre-trained weights if available.

To select which channels are retained, a TGL is plugged into the output of each convolution layer in the original architecture except the last one for preserving its generation performance. In training phase, we used ImageNet as a set of content images and chose 5 style images (i.e., $N = 5$) manually as shown in Figure 6. We trained both original and compressed model from scratch for 20K iterations with a batch size of 16. The number of pruned channels is used as a regularization factor with regularization weight $\lambda = 0.1$.

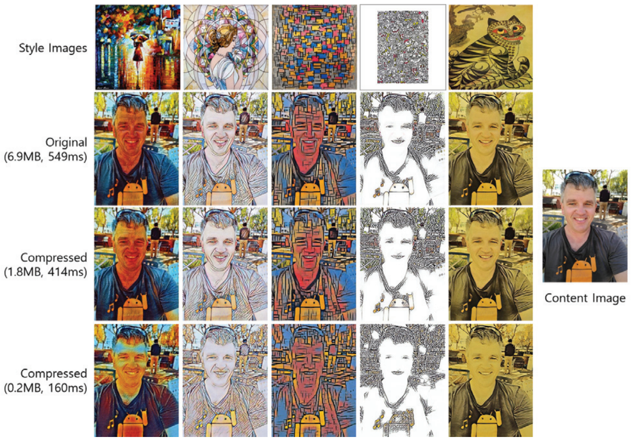


Figure 6: Results of the style transfer task. The rightmost column represents a content image and the top row represents 5 style images. *Second row*: Transferred images produced by the original model. *Third row*: Transferred images produced by the compressed model ($\rho_{ch} = 0.4$). *Fourth row*: Transferred images produced by the compressed model ($\rho_{ch} = 0.1$).

The compressed model ($\rho_{ch} = 0.1$) is ×34.5 times smaller than the original model in terms of file size (Table 2). In order to see the actual inference time, we measured the inference time on the CPU of a Galaxy S10. The compressed model ($\rho_{ch} = 0.1$) is more than ×3 times faster in terms of the inference time as shown although the generation quality preserved as shown in Figure 6.

Figure 7 shows the number of retained channels in each layer. The TGL does not select the number of pruned channels uniformly and it automatically selects which channels are pruned according to the objective function. Without the further pruning steps, our proposed method can train and prune simultaneously the model with a consideration of the target compression ratio as we mentioned in the previous section.

Optical Flow Estimation We next consider a task that learns the optical flow between images (Dosovitskiy et al. 2015; Ilg et al. 2017). In this experimentation, we used FlowNetSimple (Dosovitskiy et al. 2015), which is the same as FlowNetS[3] in (Ilg et al. 2017). FlowNetS stacks two consecutive input images together and feed them into the network to extract motion information between these images.

Starting from the pre-trained model, a TGL is plugged into the output of every convolution and deconvolution layers except the last one for preserving its generation performance. We trained the model with TGLs for 1.2M iterations with a batch size of 8. The Adam optimizer (Kingma and Ba 2015) was used with initial learning rate 0.0001 and it was halved every 200K iterations after the first 400K iterations.

[2] Source code: https://github.com/tensorflow/magenta/tree/master/magenta/models/image_stylization.

[3] Source code: https://github.com/sampepose/flownet2-tf

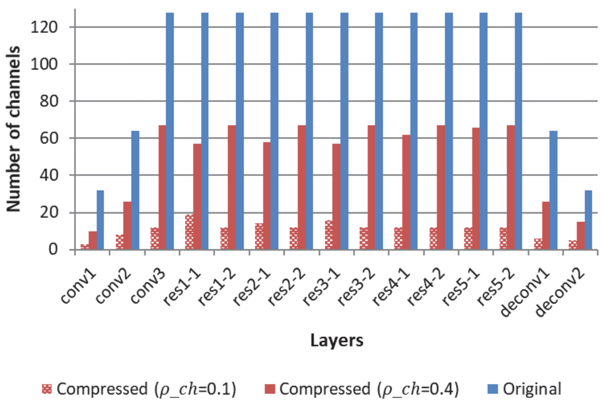


Figure 7: The number of retained channels in compressed models. Layers that should match the dimension of outputs share a single TGL (conv3, res1-2, res2-2, res3-2, res4-2, and res5-2).

Table 3: A summary of the optical flow estimation results. We measured the size of model files which are saved in binary format (.pb). For inference time, we ran both model on Intel Core i7-7700@3.60GHz CPU. The size of image is 384x512 pixels. EPE was averaged over validation data in Flying Chairs dataset.

Model	File size (MB)	EPE	Time (ms)
Original	148	3.15	292.7
$\rho_{ch} = 0.60$	54	2.91	177.8
$\rho_{ch} = 0.45$	30	3.13	155.9

As in the style transfer task, the number of pruned channels is used as a regularization factor with regularization weight $\lambda = 0.1$. We used the Flying Chairs dataset (Dosovitskiy et al. 2015) for training and testing. The performance of model is measured in average end-point-error (EPE) of validation data.

Table 3 shows the compression results. As we can see in the table, the compressed model ($\rho_{ch} = 0.45$) is $\times 4.93$ times smaller than the original model in terms of file size and more than $\times 1.88$ times faster in terms of inference time. Note that the EPE of the compressed model (3.13) is almost same with that of the original model (3.15). But it is only small a bit worse than EPE reported (2.71) in the paper (Dosovitskiy et al. 2015).

Our experimental results demonstrate that the compressed model pruned by TGL automatically finds which channels to be retained for reducing model file size, inference time, and FLOPs, while minimizing performance degradation.

Neural Machine Translation

While we have considered various applications, all of those applications are in the image domain. So as the last application we applied our pruning method to a neural machine translation task in the language domain. We tried to compress the transformer model (Vaswani et al. 2017) that has been most widely used.

Table 4: Results of pruning attention heads of a transformer model. The BLEU score measure on English-to-German newstest2014. *En* denotes the total number of retained attention heads in the encoder self-attention heads. *De* and *En-De* respectively are defined for the decoder self-attention and the encoder-decoder attention. ρ_{attn} denotes the the fraction of the total number of retained attention heads.

Model	En/De/En-De	Δ_{BLEU} (BLEU)
Original	48/48/48	– (27.32)
$\rho_{attn} = 0.83$	30/41/48	-0.06 (27.26)
$\rho_{attn} = 0.39$	11/20/25	-0.18 (27.14)

The transformer model consists of an encoder and a decoder. Each layer of the encoder has multiple self-attention heads, whereas each layer of the decoder has multiple self-attention heads and multiple encoder-decoder attention heads. To make each layer compact, we append TGFs each of which masks the corresponding attention head. Note that unlike in the previous tasks, our pruning method can prune at a block-level, an attention head, not just at the level of a single weight or channel.

We performed WMT 2014 English-to-German translation task as our benchmark and implemented on *fairseq* (Ott et al. 2019). We trained the model over 472,000 iterations from scratch. As we can see in Table 4, the BLEU score of a pruned model does not degrade much. In particular, although only 38% of attention heads are retained, the BLEU score degrades only 0.18. Our pruning method improved the computational efficiency in the language domain as well, from which we can conclude that the proposed pruning method is task-agnostic.

Conclusion

In this paper, we introduced the concept of a TGF and a differentiable pruning method as an application of the proposed TGF. The introduction of a TGF allowed us to directly optimize the loss function based on the number of parameters or FLOPs which are non-differentiable discrete values. Our proposed pruning method can be easily implemented by appending TGLs to the target layers, and the TGLs do not need additional internal parameters that need careful tuning. We showed the effectiveness of the proposed method by applying to important applications including image classification and image generation. Despite its simplicity, our experiments show that the proposed method achieves better compression results on various deep learning models. We have also shown that the proposed method is task-agnostic by performing various applications including image classification, image generation, and neural machine translation. We expect that the TGF can be applied to many more applications where we need to train discrete choices and turn them into differentiable training problems.

Acknowledgements

We would like to thank Sunghyun Choi and Haebin Shin for supports on our machine translation experiments. Kibeom Lee and Jungmin Kwon support the mobile phone experiments of the style transfer.

References

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations*.

Chollet, F., et al. 2015. Keras. https://keras.io.

Dosovitskiy, A.; Fischer, P.; Ilg, E.; Hausser, P.; Hazirbas, C.; Golkov, V.; Van Der Smagt, P.; Cremers, D.; and Brox, T. 2015. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, 2758–2766.

Dumoulin, V.; Shlens, J.; and Kudlur, M. 2017. A learned representation for artistic style. In *International Conference on Learning Representations*.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Hahn, S., and Choi, H. 2018. Gradient acceleration in activation functions. *arXiv preprint arXiv:1806.09783*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, 630–645.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, 784–800.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision (ICCV)*.

Ilg, E.; Mayer, N.; Saikia, T.; Keuper, M.; Dosovitskiy, A.; and Brox, T. 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2462–2470.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220 4598:671–80.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning filters for efficient convnets. In *International Conference on Learning Representations*.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision (ICCV)*, 2755–2763. IEEE.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: differentiable architecture search. In *International Conference on Learning Representations*.

Louizos, C.; Ullrich, K.; and Welling, M. 2017. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, 3288–3298.

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

Ott, M.; Edunov, S.; Baevski, A.; Fan, A.; Gross, S.; Ng, N.; Grangier, D.; and Auli, M. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 5998–6008.

Ye, S.; Zhang, T.; Zhang, K.; Li, J.; Xu, K.; Yang, Y.; Yu, F.; Tang, J.; Fardad, M.; Liu, S.; Chen, X.; Lin, X.; and Wang, Y. 2018. Progressive weight pruning of deep neural networks using ADMM. *arXiv preprint arXiv:1810.07378*.

Zhao, C.; Ni, B.; Zhang, J.; Zhao, Q.; Zhang, W.; and Tian, Q. 2019. Variational convolutional neural network pruning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhong, J.; Ding, G.; Guo, Y.; Han, J.; and Wang, B. 2018. Where to prune: Using lstm to guide end-to-end pruning. In *IJCAI*, 3205–3211.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.