Self-supervised Feature-Gate Coupling for Dynamic Network Pruning

Mengnan Shi\textsuperscript{a,1}, Chang Liu\textsuperscript{a,1,*}, Jianbin Jiao\textsuperscript{b}, Qixiang Ye\textsuperscript{b}

\textsuperscript{a}Tsinghua University, Beijing, 100084, China. E-mail: \{shimn2022,liuchang2022\}@tsinghua.edu.cn
\textsuperscript{b}University of Chinese Academy of Sciences, Beijing, 100049, China. E-mail: \{jiaojb,qxye\}@ucas.ac.cn

Abstract

Gating modules have been widely explored in dynamic network pruning (DNP) to reduce the run-time computational cost of deep neural networks while keeping the features representative. Despite the substantial progress, existing methods remain ignoring the consistency between feature and gate distributions, which may lead to distortion of gated features. In this paper, we propose a feature-gate coupling (FGC) approach aiming to align distributions of features and gates. FGC is a plug-and-play module that consists of two steps carried out in an iterative self-supervised manner. In the first step, FGC utilizes the $k$-Nearest Neighbor algorithm in the feature space to explore instance neighborhood relationships, which are treated as self-supervisory signals. In the second step, FGC exploits contrastive self-supervised learning (CSL) to regularize gating modules, leading to the alignment of instance neighborhood relationships within the feature and gate spaces. Experimental results validate that the proposed FGC method improves the baseline approach with significant margins, outperforming state-of-the-art methods with a better accuracy-computation trade-off. Code is publicly available at \url{github.com/smn2010/FGC-PR}.

Keywords: Contrastive Self-supervised Learning (CSL), Dynamic Network Pruning (DNP), Feature-Gate Coupling, Instance Neighborhood Relationship.

\*Corresponding author
Email address: liuchang2022@tsinghua.edu.cn (Chang Liu)
\textsuperscript{1}M. Shi and C. Liu contribute equally to this work.
1. Introduction

Convolutional neural networks (CNNs) are becoming deeper to achieve higher performance, bringing overburdened computational costs. Network pruning [1, 2], which removes the network parameters of less contribution, has been widely exploited to derive compact network models for resource-constrained scenarios. It aims to reduce the computational cost as much as possible while requiring the pruned network to preserve the representation capacity of the original network for the slightest accuracy drop.

Existing network pruning methods can be roughly categorized into static and dynamic ones. Static pruning methods [1, 3, 4], known as channel pruning, derive static simplified models by removing feature channels of negligible performance contribution. Dynamic pruning methods [2, 5, 6] derive input-dependent sub-networks to reduce run-time computational cost. Many dynamic channel pruning methods [5, 7] utilize affiliated gating modules to generate channel-wise binary masks, known as gates, which indicate the removal or preservation of channels. The gating modules explore instance-wise redundancy according to the feature variance of different inputs, i.e., channels recognizing specific features are either turned on or off for distinct input instances.

Despite substantial progresses, existing methods typically ignore the consistency between feature and gate distributions, e.g., instances with similar features but dissimilar gates, Fig. 1(upper). As gated features are produced by channel-wise multiplication of feature and gate vectors, the distribution inconsistency leads to distortion in the gated feature space. The distortion may introduce noise instances to similar instance clusters or disperse them apart, which deteriorates the representation capability of the pruned networks.

In this paper, we propose a feature-gate coupling (FGC) method to regularize the consistency between distributions of features and gates, Fig. 1(lower). The regularization is achieved by aligning neighborhood relationships of instances across feature and gate spaces. Since ground-truth annotations could not fit
Figure 1: Up: inconsistent distributions of features and gates without feature-gate coupling (FGC). Down: aligned distributions of features and gates with FGC, where instances with similar semantics would also have similar gates and vice versa. The alignment indicates that FGC allocates consistent sub-networks with specific representation capacities for semantically similar instances, which reduces the representation redundancy and thereby improves the pruning effects.

the variation of neighborhood relationships in feature spaces across different semantic levels, we propose to fulfill the feature-gate distribution alignment in a self-supervised manner.

Specifically, FGC consists of two steps, which are conducted iteratively. In the first step, FGC utilizes the $k$-Nearest Neighbor ($k$NN) algorithm to explore instance neighborhood relationships in the feature space, where the nearest neighbors of each instance are identified by evaluating feature similarities. The explored neighborhood relationships are then used as self-supervisory signals in the second step, where FGC utilizes contrastive self-supervised learning (CSL) \[\text{CSL}\] to regularize gating modules. The nearest neighbors of each instance are identified as positive instances while others as negative instances when defining the contrastive loss function. The contrastive loss is optimized in the gate space to pull positive/similar instances together while pushing negative/dissimilar instances away from each other, which guarantees consistent
instance neighborhood relationships across the feature and gate spaces. With gradient back-propagation, parameters in the neural networks are updated as well as feature and gate distributions. FGC then goes back to the first step and conducts the method iteratively until convergence. As been designed to perform in the feature and gate spaces, FGC is architecture agnostic and can be applied to various dynamic channel pruning methods in a plug-and-play fashion. The contributions of this study are summarized as follows:

1) We propose a plug-and-play feature-gate coupling (FGC) method to alleviate the distribution inconsistency between features and corresponding gates of dynamic channel pruning methods in a self-supervised manner.

2) We implement FGC by iteratively performing neighborhood relationship exploration and feature-gate alignment, which aligns instance neighborhood relationships via a designed contrastive loss.

3) We conduct experiments on public datasets and network architectures, justifying the efficacy of FGC with significant DNP performance gains.

2. Related Works

Network pruning methods can be coarsely classified into static and dynamic ones. Static pruning methods prune neural networks by permanently removing redundant components, such as weights \([3, 9, 10]\), channels \([1, 11, 12, 13]\) or connections \([14]\). They aim to reduce shared redundancy for all instances and derive static compact networks. Dynamic pruning methods reduce network redundancy related to each input instance, which usually outperform static methods and have been the research focus of the community.

**Dynamic network pruning (DNP)** \([2, 5, 6]\) in recent years has exploited the instance-wise redundancy to accelerate deep networks. The instance-wise redundancy could be roughly categorized into spatial redundancy, complexity redundancy, and parallel representation redundancy.
Spatial redundancy commonly exists in feature maps where objects may lay in limited regions of the images, while the large background regions contribute marginally to the prediction result. Recent methods \cite{6, 15} deal with spatial redundancy to reduce repetitive convolutional computation, or reduce spatial-wise computation in tasks where the spatial redundancy issue becomes more severe due to superfluous backgrounds, such as object detection \cite{10}, etc.

The complexity redundancy has recently deserved attention. As CNNs are getting deeper for recognizing “harder” instances, an intuitive way to reduce computational cost is executing fewer layers for “easier” instances. Networks could stop at shallow layers \cite{17} once classifiers are capable of recognizing input instances with high confidence. However, the continuous layer removal could be a sub-optimal scheme, especially for ResNet \cite{18} where residual blocks could be safely skipped \cite{2, 19} without affecting the inference procedure.

Representation redundancy exists in the network because parallel components, \textit{e.g.}, branches and channels, are trained to recognize specific patterns. To reduce such redundancy, previous methods \cite{20, 21} derive dynamic networks using policy networks or neural architecture search methods. However, they introduce additional overhead and require special training procedures, \textit{i.e.}, reinforcement learning, or neural architecture search. Recently, researchers explore pruning channels with affiliated gating modules \cite{5, 7}, which are small networks with negligible computational overhead. The affiliated gating modules perform the channel-wise selection by gates, reducing redundancy and improving compactness. Instead of training by only classification loss, \cite{22} defines a self-supervised task to train gating modules, the target masks of which are generated according to corresponding features. Despite the progress in the reviewed channel pruning methods, the correspondence between gates and features remains ignored, which is the research focus of this study.

Contrastive self-supervised learning (CSL) \cite{23} is once used to keep consistent embeddings across spaces, \textit{i.e.}, when a set of high dimensional data points are mapped onto a low dimensional manifold, the “similar” points in the original space are mapped to nearby points on the manifold. It achieves
Figure 2: Flowchart of the proposed feature-gate coupling (FGC) method for DNP. FGC consists of two iterative steps: neighborhood relationship exploration for instance-wise neighborhood relationship modeling and feature-gate alignment for gating module regularization.

this goal by moving similar pairs together and separating dissimilar pairs in the low dimensional manifold, where “similar” or “dissimilar” points are deemed according to prior knowledge in the high dimensional space.

CSL has been widely explored in unsupervised and self-supervised learning \cite{8, 24, 25} to learn consistent representation. The key point is to identify similar/positive or dissimilar/negative instances. An intuitive method is taking each instance itself as positive \cite{8}. However, its relationship with nearby instances is ignored, which is solved by exploring the neighborhood \cite{24} or local aggregation \cite{26}. Unsupervised clustering methods \cite{27, 28} further enlarge the positive sets by treating instances in the same cluster as positives. It is also noticed that positive instances could be gained from multiple views \cite{29} of a scene. Besides, recent methods \cite{25} generate positive instances by siamese encoders, which process data-augmentation of original instances. The assumption is that the instance should stay close in the representation space after whatever augmentation is applied.

According to previous works, the contrastive loss is an ideal tool to align distributions or learn consistent representation. Therefore, we develop a CSL method in our work, \textit{i.e.}, the positives are deemed by neighborhood relation-
ships, and the loss function is defined as a variant of InfoNCE \cite{30}.

3. Methodology

Inspired by InstDisc \cite{8}, that visually similar instances are typically close in the feature space, we delve into keeping the distribution alignment between features and gates to avoid distortion after pruning. To this end, we explore instance neighborhood relationships within the feature space for self-supervisory signals and propose the contrastive loss to regularize gating modules using the generated signals, Fig. 2. The two iterative steps can be summarized as follows:

- **Neighborhood relationship exploration:** We present a $k$NN-based method to explore instance neighborhood relationships in the feature space. Specifically, we evaluate cosine similarities between feature vectors of instance pairs. The similarities are then leveraged to identify the closest instances in the feature space, denoted as nearest neighbors. Finally, indexes of those neighbors are delivered to the next step.

- **Feature-gate alignment:** We present a CSL-based method to regularize gating modules by defining positive instances with indexes of neighbors from the previous step. The contrastive loss is then minimized to pull the positives together and push negatives away in the gate space, which achieves feature-gate distribution alignment by improving the consistency between semantic relationships of instances and corresponding gate similarities.

The two steps are conducted iteratively until the optimal dynamic model is obtained. With the potential to be applied to other DNP methods, our FGC is inserted into the popular gating module \cite{7} as an instance.

3.1. Preliminaries

We introduce the basic formulation of gated convolutional layers. For an instance in the dataset $\{x_i, y_i\}_{n=1}^N$, its feature in the $l$-th layer of CNNs is
denoted as:

\[ f^l_i = F(x^l_{i-1}) = \text{ReLU}(w^l \ast x^l_{i-1}). \]  

(1)

where \( x^l_{i-1} \in \mathbb{R}^{C_{l-1} \times W_l \times H_l} \) and \( f^l_i \in \mathbb{R}^{C_l \times W_l \times H_l} \) are respectively the input and output of the \( l \)-th convolutional layer. \( C^l \) is the channel number. \( W^l \) and \( H^l \) represent the width and height of the feature map. \( w^l \) denotes the convolutional weight matrix. \( \ast \) is the convolution operator, and ReLU is the activation function. Batch normalization is applied after each convolution layer.

Dynamic channel pruning methods temporarily remove channels to reduce related computation in both preceding and succeeding layers, where a binary gate vector, generated from the preceding feature \( x^l_{i-1} \), is used to turn on or off corresponding channels. The gated feature in the \( l \)-th layer is denoted as:

\[ x^l_i = g^l_i \odot f^l_i, \]  

(2)

where \( g^l_i \) denotes the gate vector and \( \odot \) denotes the channel-wise multiplication.

The gate vectors come from various gating modules with uniform functionality. Without loss of generality, we use the form of gating module [7], which is defined as:

\[ g^l_i = G(x^l_{i-1}) = B(M(p(x^l_{i-1}))). \]  

(3)

The gating module first goes with the global average pooling layer \( p \), which is used to generate the spatial descriptor of the input feature \( x^l_{i-1} \). A lightweight network \( M \) with two fully connected layers is then applied to generate the gating probability \( \pi^l_i = \text{Linear}(p(x^l_{i-1})) \). Finally, the BinConcrete activation function \( B \) is used to sample from \( \pi^l_i \) to get the binary mask \( g^l_i \).

3.2. Implementation

**Neighborhood Relationship Exploration.** Instance neighborhood relationships vary in feature spaces with different semantic levels, e.g., in a low-level feature space, instances with similar colors or texture may be closer, while in a high-level feature space, instances from the same class may gather together. Thus, human annotations could not provide adaptive supervision of
instance neighborhood relationships across different network stages. We propose to utilize the kNN algorithm to adaptively explore instance neighborhood relationships in each feature space and extract self-supervision accordingly for feature-gate distribution alignment.

Note that the exploration could be adopted in arbitrary layers, e.g., the $l$-th layer of CNNs. In practice, in the $l$-th layer, the feature $\hat{f}^l_i$ of the $i$-th instance is first fed to a global average pooling layer, as $\hat{f}^l_i = p(f^l_i)$. The dimension of pooled feature reduces from $C^l \times W^l \times H^l$ to $C^l \times 1 \times 1$, which is more efficient for processing. We then compute its similarities with other instances in the training dataset as:

$$S^l[i, j] = \hat{f}^l_i \cdot \hat{f}^l_j,$$

(4)

where $S^l \in \mathbb{R}^{N \times N}$ denotes the similarity matrix and $S^l[i, j]$ denotes similarity between the $i$-th instance and the $j$-th instance. $N$ denotes the number of total training instances. To identify nearest neighbors of $x_i$, we sort the $i$-th row of $S_l$ and return the subscripts of the $k$ biggest elements, as

$$N^l_i = \text{topk}(S^l[i,:]).$$

(5)

The set of subscripts $N^l_i$, i.e., indexes of neighbor instances in the training dataset, are treated as self-supervisory signals to regularizing gating modules.

To compute the similarity matrix $S^l$ in Eq. (4), $\hat{f}^l_j$ for all instances are required. Instead of exhaustively computing these pooled features at each iteration, we maintain a feature memory bank $V^l_f \in \mathbb{R}^{N \times D}$ to store them [8]. $D$ is the dimension of the pooled feature, i.e., the number of channels $C^l$. Now for the $i$-th pooled feature $\hat{f}^l_i$, we compute its similarities with vectors in $V^l_f$. After that, the memory bank is updated in a momentum way, as

$$V^l_f[i] \leftarrow m \cdot V^l_f[i] + (1 - m) \cdot \hat{f}^l_i,$$

(6)

where the momentum coefficient is empirically set to 0.5. All vectors in the memory bank are initialized as unit random vectors.

**Feature-Gate Alignment.** We identify the nearest neighbor set $\{x_j, j \in N^l_i\}$ of each instance $x_i$ in the feature space. Thereby, instances in each neighbor
set are treated as positives to be pulled closer while others as negatives to be pushed away in the gate space. To this end, we utilize the contrastive loss in a self-supervised manner.

Specifically, by defining nearest neighbors of an instance as its positives, the probability of an input instance \( x_j \) being recognized as nearest neighbors of \( x_i \) in the gate space, as

\[
p(j|\pi^l_i) = \frac{\exp(\pi^T_j \cdot \pi^l_i / \tau)}{\sum_{k=1}^{N} (\exp(\pi^T_k \cdot \pi^l_i) / \tau)}
\]

where \( \pi^l_i \) and \( \pi^l_j \) are gating probabilities, the gating module outputs of \( x_i \) and \( x_j \) respectively. \( \tau \) is a temperature hyper-parameter and set as 0.07 according to [8]. Then the loss function is derived after summing up \( p(j|\pi^l_i) \) of all \( k \) neighbors, as

\[
L_g^l = - \sum_{j \in N^l_i} \log(p(j|\pi^l_i)).
\]

Since nearest neighbors are defined as positives and others as negatives, Eq. (8) is minimized to draw previous nearest neighbors close, i.e., reproducing the instance neighborhood relationships within the gate space. We maintain a gate memory bank \( V^l_g \in \mathbb{R}^{N \times D} \) to store gating probabilities. For each \( \pi^l_i \), its positives are taken from \( V^l_g \) according to \( N^l_i \). After computing Eq. (8), \( V^l_g \) is updated in a similar momentum way as updating the feature memory bank. All vectors in \( V^l_g \) are initialized as unit random vectors.

**Overall Objective Loss.** Besides the proposed contrastive loss, we utilize the cross-entropy loss (\( L_{ce} \)) for the classification task and the \( L_0 \) loss to introduce sparsity. \( L_0^l = ||\pi^l_i||_0 \) is used in each gated layer, where \( ||\cdot||_0 \) is \( L_0 \)-norm.

The overall loss function of our method can be formulated as:

\[
\text{loss} = L_{ce} + \eta \cdot \sum_{l \in \Omega} L_g^l + \rho \cdot \sum_{l} L_0^l,
\]

where \( L_g^l \) denotes the contrastive loss adopted in the \( l \)-th layer and \( \eta \) the corresponding coefficient, \( \Omega \) denotes the layers \( L_g^l \) applied in. The coefficient \( \rho \) controls the sparsity of pruned models, i.e., a bigger \( \rho \) leads to a sparser model with lower accuracy, while a lower \( \rho \) derives less sparsity and higher accuracy.
Algorithm 1 Feature-Gate Coupling.

Input: Training instances $x_i$, labels $y_i$ of an imagery dataset;

Output: Dynamic network with parameters $\theta$;

1: Initialize feature and gate memory banks $V^l_f$ and $V^l_g$ for layers in $\Omega$;
2: for layer $l = 1$ to $L$ do
3: Generate feature $f_i$ and gating probability $\pi_i$ of instance $x_i$;
4: if $l \in \Omega$ then
5: Generate pooled feature $\hat{f}_i$ of instance $x_i$;

Step 1. Neighborhood Relationship Exploration
6: Calculate the similarities of $\hat{f}_i$ with $V^l_f[j], (j \in (1, N), j \neq i)$, (Eq. (4))
7: Get indexes of nearest neighbors $N^l_i$, (Eq. (5));
8: Update the feature memory bank $V^l_f$, (Eq. (6));

Step 2. Feature-Gate Alignment
9: Draw positive instances w.r.t. $N^l_i$ from the gate memory bank $V^l_g$;
10: Calculate the contrastive loss in the gate space $L^l_g$, (Eq. (8));
11: Update the gate memory bank $V^l_g$;
12: end if
13: Calculate $L^l_0$;
14: end for
15: Calculate the overall objective loss, (Eq. (9));
16: Updating parameters $\theta$ by gradient back-propagation.

The learning procedure of FGC is summarized as Algorithm 1.

3.3. Theoretical Analysis

We analyze the plausibility of designed contrastive loss (Eq. (8)) from the perspective of mutual information maximization. According to [30], the InfoNCE loss is used to maximize the mutual information between learned representation and input signals. It is assumed that minimizing the contrastive loss leads to the improved mutual information between features and corresponding gates.
We define the mutual information between gates and features as:

\[
I(f^l, g^l) = \sum_{f^l, g^l} p(f^l, g^l) \log \frac{p(f^l|g^l)}{p(f^l)},
\]

where the density ratio based on \( f^l_j \) and \( g^l_i \) is proportional to a positive real score as \( q(f^l_j, g^l_i) \propto p(f^l_j|g^l_i)/p(f^l_j) \) [10], which is \( q(f^l_j, g^l_i) = \exp(\pi^T_j \cdot \pi^l_i/\tau) \) in our case. \( \pi^l_i \) is the gating probabilities for \( g^l_i \) and \( \pi^l_j \) belongs to its positives. The InfoNCE loss item could be rewritten as:

\[
\mathcal{L}^l_g = -\mathbb{E}_X \left[ \log \frac{q(f^l_j, g^l_i)}{\sum_{f^l_j \in f^l_k} q(f^l_k, g^l_i)} \right].
\]

By taking the density ratio into the equation, we could derive the relationship between InfoNCE loss and mutual information, where the superscript \( l \) is omitted for brevity,

\[
\mathcal{L}_{\text{NCE}}^l = -\mathbb{E}_X \log \left[ \frac{q(f^l_j, g^l_i)}{\sum_{f^l_k \in f^l_j} q(f^l_k, g^l_i)} \right]
\]

\[
= -\mathbb{E}_X \log \left[ \frac{p(f^l_j|g^l_i)}{p(f^l_j)} + \sum_{f^l_k \in X_{\text{neg}}} \frac{p(f^l_k|g^l_i)}{p(f^l_k)} \right]
\]

\[
= \mathbb{E}_X \log \left[ 1 + \frac{p(f^l_j)}{p(f^l_j|g^l_i)} \sum_{f^l_k \in X_{\text{neg}}} \frac{p(f^l_k|g^l_i)}{p(f^l_k)} \right]
\]

\[
\approx \mathbb{E}_X \log \left[ 1 + \frac{p(f^l_j)}{p(f^l_j|g^l_i)} (N-1) \mathbb{E}_{f^l_k \in f^l_j} \frac{p(f^l_k|g^l_i)}{p(f^l_k)} \right]
\]

\[
= \mathbb{E}_X \log \left[ 1 + \frac{p(f^l_j)}{p(f^l_j|g^l_i)} (N-1) \right]
\]

\[
= \mathbb{E}_X \log \left[ \frac{p(f^l_j|g^l_i) - p(f^l_j)}{p(f^l_j|g^l_i)} + \frac{p(f^l_j)}{p(f^l_j|g^l_i)} N \right]
\]

\[
\geq \mathbb{E}_X \log \left[ \frac{p(f^l_j)}{p(f^l_j|g^l_i)} N \right]
\]

\[
= -I(f^l_j, g^l_i) + \log(N),
\]

where \( N \) is the number of training samples, and \( X_{\text{neg}} \) denotes the set containing negative instances. The inequality holds because \( f^l_j \) and \( g^l_i \) is not independent and \( p(f^l_j|g^l_i) \geq p(f^l_j) \).

As \( I(f^l_j, g^l_i) \geq \log(N) - \mathcal{L}_{\text{NCE}}^l \), we could conclude that minimizing the InfoNCE loss item would maximize a lower bound on mutual information, justi-
fying the efficacy of proposed contrastive loss $\mathcal{L}_g = \sum_{j \in \mathcal{N}_i} \mathcal{L}_{NCE}^j$. Accordingly, the improvement of mutual information between features and corresponding gates facilities distribution alignment. (Please refer to Fig. 10)

4. Experiments

In this section, the experimental settings for dynamic channel pruning are first described. Ablation studies are then conducted to validate the effectiveness of FGC. The alignment of instance neighborhood relationships by the FGC is also quantitatively and qualitatively analyzed. We show the performance of learned dynamic networks on image classification benchmark datasets and compare them with state-of-the-art pruning methods. Finally, We conduct object detection and semantic segmentation experiments to show the generation ability of FGC.

4.1. Experimental Settings

Datasets. Experiments on image classification are conducted on benchmarks including CIFAR10/100 [32] and ImageNet [33]. CIFAR10 and CIFAR100 consist of 10 and 100 categories, respectively. Both datasets contain 60K colored images, 50K and 10K of which are used for training and testing. ImageNet is an important large-scale dataset composed of 1000 object categories, which has 1.28 million images for training and 50K images for validation.

We conduct object detection on PASCAL VOC 2007 [34], the trainval and test sets of which are used for training and testing, respectively. Instance segmentation is conducted on Cityscapes [35]. We use the train set for training and the validation set for testing.

Implementation Details. For CIFAR10 and CIFAR100, training images are applied with standard data augmentation schemes [7]. We use the ResNet of various depths, i.e., ResNet–{20, 32, 56}, to fully demonstrate the effectiveness of our method. On CIFAR10, models are trained for 400 epochs with a minibatch of 256. A Nesterov’s accelerated gradient descent optimizer is used, where
the momentum is set to 0.9 and the weight decay is set to 5e-4. It is noticed that no weight decay is applied to parameters in gating modules. As for the learning rate, we adopt a multiple-step scheduler, where the initial value is set to 0.1 and decays by 0.1 at milestone epochs [200, 275, 350]. We also evaluate FGC on WideResNet [35] and MobileNetV2 [37] (a type of lightweight network). Specifically, models are trained for 200 epochs with a mini-batch of 256. The initial learning rate is 0.1 and decays by 0.2 at epochs [60, 120, 160]. On CIFAR100, models are trained for 200 epochs with a mini-batch of 128. The same optimization settings as CIFAR10 are used. The initial learning rate is set to 0.1, then decays by 0.2 at epochs [60, 120, 160]. For ImageNet, the data augmentation scheme is adopted from [18]. We only use ResNet-18 to validate the effectiveness on the large-scale dataset due to time-consuming training procedures. We train models for 130 epochs with a mini-batch of 256. A similar optimizer is used except for the weight decay of 1e-4. The learning rate is initialized to 0.1 and decays by 0.1 at epochs [40, 70, 100]. FGC is employed in the last few residual blocks of ResNets, i.e., the last two blocks of ResNet-\{20,32\}, the last four blocks of ResNet-56, and the last block of ResNet-18. Hyper-parameters $\eta$ and $k$ are set to 0.003 and 200 by default, respectively. We set the l0 loss coefficient $\rho$ as 0.4 to introduce considerable sparsity into gates.

For PASCAL VOC 2007, we use the detector and the data augmentation described in Faster R-CNN [38]. The Faster R-CNN model consists of the ResNet backbone and Feature Pyramid Network [39] (FPN). Specifically, We replace the backbone with the gated ResNet-\{34, 50\} pre-trained on ImageNet, leaving other components unchanged. We fine-tune the detector under the 1x schedule where the initial learning rate is 0.001, the momentum is 0.9, and the weight decay is 1e-4. No weight decay is applied in gating modules. FGC is employed in the last stage of the backbone. During the fine-tuning process, hyper-parameters $\eta$ and $k$ are set to 0.0001 and 1, respectively.

For Cityscapes, we follow the data augmentation and the method as PSP-Net [40]. PSPNet with a pre-trained ResNet-50 backbone is trained with a
Table 1: Test errors and pruning ratios of the pruned ResNet-20 w.r.t the number of nearest neighbors $k$. “Error” is the classification error, where “↓” denotes the lower the better. “Pruning” is the pruning ratio of computation, where “↑” denotes the higher the better. (Ditto for other tables.)

| $k$  | 5  | 20 | 100 | 200 | 512 | 1024 | 2048 | 4096 |
|------|----|----|-----|-----|-----|------|------|------|
| Error (%) ↓ | 8.47 | 8.01 | 8.00 | 7.91 | 8.16 | 8.04 | 8.09 | 8.39 |
| Pruning (%) ↑ | 50.1 | 54.1 | 53.4 | 55.1 | 53.3 | 54.9 | 53.0 | 53.8 |

Figure 3: Test accuracies and computation of the pruned ResNet-20 w.r.t the number of nearest neighbors $k$.

mini-batch of 6 for 300 epochs. The optimizer is an SGD optimizer with the poly learning rate schedule, where the initial learning rate is 0.02 and the decay exponent is 0.9. The weight decay and momentum are 5e-4 and 0.9, respectively. The weight decay of gating modules is 2e-6. In practice, We employ FGC in the last stage of the backbone. We set $\eta$ and $k$ to 0.0001 and 4.

4.2. Ablation studies

We conduct ablation studies on CIFAR10 to validate the effectiveness of our method by exploring hyper-parameters, residual blocks, neighborhood relationships, and accuracy-computation trade-offs.
Table 2: Test errors and pruning ratios of the pruned ResNet-20 w.r.t coefficient $\eta$.

| $\eta$ ($\times 1e-3$) | 0    | 0.5  | 1    | 1.5  | 2    | 2.5  | 3    |
|------------------------|------|------|------|------|------|------|------|
| Error (%) ↓            | 8.59 | 8.47 | 8.22 | 8.14 | 8.12 | 8.11 | 7.91 |
| Pruning (%) ↑          | 5.40 | 54.8 | 53.2 | 53.9 | 55.3 | 54.8 | 55.1 |

| $\eta$ ($\times 1e-3$) | 3.5  | 4    | 4.5  | 5    | 10   | 20   |
|------------------------|------|------|------|------|------|------|
| Error (%) ↓            | 8.07 | 8.04 | 8.18 | 7.94 | 8.01 | 8.21 |
| Pruning (%) ↑          | 54.9 | 54.5 | 53.7 | 53.1 | 53.0 | 52.4 |

**Hyper-parameters of Contrastive Loss.** In Table 1 and Fig. 3, we show the performance of pruned models given different $k$, i.e., the number of nearest neighbors. For a fair comparison, the computation of each model is tuned to be roughly the same so that it can be evaluated by accuracy. As shown by Fig. 3, FGC is robust about $k$ in a wide range, i.e., the accuracy reaches the highest value around $k=200$, and keeps stable when $k$ falls into the range (20, 1024).

In Table 2 and Fig. 4, we show the performance of pruned models under different $\eta$, i.e., the loss coefficient. Similarly, the computation of each model is also tuned to be almost the same. The contrastive loss brings a significant accuracy improvement compared with models without FGC. With the increase of $\eta$, the accuracy improves from 91.4% to 92.09%. The best performance is achieved when $\eta = 0.003$. While accuracy slightly drops when $\eta$ is larger than 0.01, it remains better than that at $\eta = 0$.

**Residual Blocks.** In Table 3, we exploit whether FGC is more effective when employed in multiple residual blocks. The first row of Table 3 denotes models trained without FGC. After gradually removing shallow residual blocks from adopting FGC, experimental results show that FGC in shallow stages brings a limited performance gain despite more memory banks, e.g., total 18 memory banks used when employing FGC in all blocks. However, FGC in the last stage, especially from the 8-th to the 9-th blocks, contributes better to the final performance, i.e., the lowest error (7.91%) with considerable computation reduction (55.1%). It is concluded that the residual blocks in deep stages benefit more from FGC, probably due to more discriminative features and more
conditional gates. Therefore, taking both training efficiency and performance into consideration, we employ FGC in the deep stages of ResNets.

**Neighborhood Relationship Sources.** In Table 4, we validate the necessity of exploring instance neighborhood relationships in feature spaces by comparing the counterparts. Specifically, we randomly select $k$ instances with the same label as neighbors for each input instance, denoted as “Label”; we also utilize $k$NN in the gate space to generate nearest neighbors, denoted as “Gate”. Moreover, since FGC is applied in multiple layers separately, an alternative adopts neighborhood relationships from the same layer, e.g., the last layer. We define this case as “Feature” with “Shared” instance neighborhood relationships.

Results in Table 4 demonstrate that our method, i.e., “Feature” with independent neighbors, prevails over all the counterparts, which fail to explore proper instance neighborhood relationships for alignment. For example, labels are too coarse or discrete to describe neighborhood relationships. Exploring neighbors in the gate space enhances its misalignment with the feature space.
Table 3: Test errors and pruning ratios of the pruned ResNet-20 when employing FGC in different blocks. For ResNet-20, which consists of three stages, each of them comprising three basic residual blocks, we use indexes of “Stages” and “Blocks” to locate the application of FGC.

| Stages | Blocks | Memory Banks | Error (%) ↓ | Pruning (%) ↑ |
|--------|--------|--------------|-------------|--------------|
| -      | -      | -            | 8.49        | 54.3         |
| 1\(^{st}\) to 3\(^{rd}\) | 1\(^{st}\) to 9\(^{th}\) | 18           | 8.08        | 53.4         |
| 2\(^{nd}\) to 3\(^{rd}\) | 4\(^{th}\) to 9\(^{th}\) | 12           | 8.36        | **56.0**     |
| 3\(^{rd}\) | 7\(^{th}\) to 9\(^{th}\) | 6            | 8.21        | **56.0**     |
|        | 8\(^{th}\) to 9\(^{th}\) | 4            | **7.91**    | 55.1         |
|        | Only 9\(^{th}\) | 2            | 8.14        | 54.5         |

Table 4: Comparisons of neighbor sources with ResNet-20 for instance neighborhood relationship exploration.

| Neighbor Source | Shared | Error (%) ↓ | Pruning (%) ↑ |
|-----------------|--------|-------------|--------------|
| -               | -      | 8.49        | 54.3         |
| Label           | ✓      | 8.16        | 52.0         |
| Gate            | ✓      | 8.50        | 51.6         |
| Feature         | ✓      | 8.14        | 51.2         |
| Feature         | ✓      | **7.91**    | **55.1**     |

Therefore the performance, either error or pruning ratio (8.50%, 51.6%), is even worse than the model without FGC. Moreover, “Shared” neighbors from the same feature space could not deal with variants of intermediate features, thereby limiting the improvement.

We conclude that the proposed FGC consistently outperforms the compared methods, which attributes to the alignment of instance neighborhood relationships between feature and gate spaces in multiple layers.

**Computation-Accuracy Trade-offs.** In Fig. 5, we show the computation-accuracy trade-offs of FGC and the baseline method by selecting \( L_0 \) loss coefficient \( \rho \) from values of \{0.01, 0.02, 0.05, 0.1, 0.2, 0.4\}. Generally speaking, the accuracy of pruned models increases as the total computational cost increases. Compared with pruned models without FGC, the pruned ResNet-32 with FGC
Figure 5: Computation-accuracy trade-offs of pruned ResNet-{20, 32}. The “w/” and “w/o” refer to “with” and “without”, respectively, indicating whether the FGC method is used or not. (Ditto for other tables or figures.)

obtains higher accuracy with lower computational cost, as shown in the top right of Fig. 5. At the left bottom of Fig. 5 the pruned ResNet-20 with FGC achieves 0.5% higher accuracy using the same computational cost. The efficacy of FGC is clarified that increasing performances compared with the baseline models can be observed at lower pruning ratios.

4.3. Analysis

Visualization Analysis. In Fig. 6 we use UMAP to visualize the correspondence of features and gates during the training procedure. Specifically, we show features and gates from the last residual block (conv3-3) of ResNet-20, where features are trained to be discriminative, i.e., easy to be distinguished in the embedding space. It’s seen that features and gates are simultaneously optimized and evolve together during training. In the last epoch, as instance neighborhood relationships have been well reproduced in the gate space, we can observe gates with a similar distribution as features, e.g., instances of the same
clusters in the feature space are also assembled in the gate space. Intuitively, the coupling of features and gates promises better predictions of our pruned models with given pruning ratios.

In Fig. 7, we compare feature and gate embeddings with pruned models in BAS [7]. Specifically, they are drawn from the last residual block (conv3-3) of ResNet-20 and the second to last residual block (conv3-4) of ResNet-32. As shown, our method brings pruned models with coupled features and gates, compared with BAS. For example, “bird” (green), “frog” (grey) and “deer” (brown) instances entangle together (left bottom of Fig. 7a), while adopting FGC (right bottom of Fig. 7a), they are clearly distinguished as corresponding features. Similar observations are seen in Fig. 7b, validating the alignment effect of FGC, which facilitates the model to leverage specific inference paths for semantically
consistent instances and thereby decrease representation redundancy.

In Fig. 8, we visualize the execution frequency of channels in pruned ResNet-20 for each category on CIFAR10. In shallow layers (e.g., conv1-3), gates almost entirely turn on or off for arbitrary categories, indicating why FGC is unnecessary: a common channel compression scheme is enough. However, in deep layers (e.g., conv3-3), dynamic pruning is performed according to the semantics of categories. It’s noticed that semantically similar categories, such as “dog” and “cat”, also share similar gate execution patterns. We conclude that FGC utilizes and enhances the correlation between instance semantics and pruning patterns to pursue better representation.

In Fig. 9, we show images in the ordering of gate similarities with the target ones (leftmost). One can see that the semantic similarities and gate similarities of image instances decrease consistently from left to right.

**Mutual Information Analysis.** In Fig. 10 we validate that FGC improves the mutual information between features and gates to facility distribution alignment, as discussed in section 3.3. For comparison, only partial deep layers are employed with FGC in our pruned models, indicated by “∗”. Specifically, the normalized mutual information (NMI) between features and gates is roughly unchanged for layers not employed with FGC. However, we observe the consistent improvement of NMI between features and gates for layers employed with
Figure 9: Sorted images w.r.t gate similarities on ImageNet validation set. In each row, we list example images according to their gate similarities (values above images) with the leftmost one. Gates are drawn from the 8-th gated block of ResNet-18.

Figure 10: Normalized mutual information (NMI) in deep residual blocks of ResNet-\{20, 32, 56\}, compared with pruned models in BAS [7]. “Feature” denotes NMI between features and labels; “Gate” denotes NMI between gates and labels; “Feature-Gate” denotes NMI between features and gates. “∗” denotes residual blocks employed with FGC.

FGC compared with the pruned models in BAS [7].

4.4. Performance

We compare the proposed method with state-of-the-art methods, including static and dynamic methods. For a clear and comprehensive comparison, we classify these methods into different types w.r.t. pruned objects. Specifically, most static methods rely on pruning channels (or pruning filters, which is equivalent), while some prune weights in filter kernels or connections between channels. Dynamic methods are classified into three types, layer pruning, spatial prun-
ing, and channel pruning, as discussed in Sec. 2. The performance is evaluated by two metrics, classification errors and average pruning ratios of computation. Higher pruning ratios and lower errors indicate better-pruned models.

**CIFAR10.** In Table 5, comparisons with SOTA methods to prune ResNet-{20, 32, 56} are shown. FGC significantly outperforms static pruning methods. For example, our dynamic model for ResNet-56 reduces 66.2% computation, 11.2% more than RL-MCTS [46], given a rather similar accuracy (6.41% vs 6.44%). The reason lies in that FGC further reduces the instance-wise redundancy of input samples.

It prevails various types of dynamic pruning techniques. Specifically, FGC significantly outperforms AIG [19] and Blockdrop [2], typical dynamic layer pruning methods that derive dynamic inference by skipping layers. AIG for ResNet-20 achieves a slightly higher error than FGC but with less than half of the computational reduction. BlockDrop for ResNet-32 prunes 54.0% computation with 8.70% error, 1.44% higher than FGC for ResNet-32. Unlike layer pruning, FGC excavates redundancy of each gated layer in a fine-grained manner, maximally reducing irrelevant computation with minor representation degradation. FGC also consistently outperforms dynamic spatial pruning methods (SACT [16] and DynConv [43]), which typically ignore the distribution consistency between instances and pruned architectures, by a large margin. FGC prunes similar computational costs with lower errors than SACT for ResNet-20 (53.7% reduction and 10.02% error). Compared with DynConv for ResNet-32 (51.3% reduction and 7.47% error), FGC prunes much more computation with slightly lower error.

We further show that FGC achieves a superior accuracy-computation trade-off than BAS [7] and FBS [5], the very relevant dynamic channel pruning methods with gating modules. For ResNet-56, FGC prunes 3.3% more computation than BAS with approximate classification error. For ResNet-20, FGC prunes 3.1% more computation with lower error than BAS. FGC alleviates the inconsistency between distributions of features and gates in these methods. Our superior results validate the effectiveness of feature regularization toward gating
Table 5: Comparisons of the pruned ResNet on CIFAR-10. Pruning methods are classified into different types w.r.t. pruned objects, i.e., “C” for channels, “CN” for connections, “W” for weights, “L” for layers, “S” for spatial. (Ditto for other tables.)

| Model  | Method  | Dynamic | Type | Error (%) ↓ | Pruning (%) ↑ |
|--------|---------|---------|------|-------------|---------------|
| ResNet-20 | Baseline | -       | -    | 7.22        | 0.0           |
|         | SFP [42]       | ×       | C    | 9.17        | 42.2          |
|         | FPGM [1]       | ×       | C    | 9.56        | 54.0          |
|         | DSA [14]       | ×       | CN   | 8.62        | 50.3          |
|         | Hinge [11]     | ×       | C    | 8.16        | 45.5          |
|         | DHP [3]        | ×       | W    | 8.46        | 51.8          |
|         | AIG [19]       | ✓       | L    | 8.49        | 24.7          |
|         | SACT [16]      | ✓       | S    | 10.02       | 53.7          |
|         | DynConv [43]   | ✓       | S    | 8.10        | 50.7          |
|         | FBS [5]        | ✓       | C    | 9.03        | 53.1          |
|         | BAS [7]        | ✓       | C    | 8.49        | 54.3          |
| FGC (Ours) | ✓       | C    | 7.01 | 55.1        |
| FGC (Ours) | ✓       | C    | 8.13 | 57.4        |
| ResNet-32 | Baseline | -       | -    | 6.75        | 0.0           |
|         | SFP [42]       | ×       | C    | 7.92        | 41.5          |
|         | FPGM [1]       | ×       | C    | 8.07        | 53.2          |
|         | AIG [19]       | ✓       | L    | 8.49        | 24.7          |
|         | BlockDrop [2]  | ✓       | L    | 8.70        | 54.0          |
|         | SACT [16]      | ✓       | S    | 8.00        | 38.4          |
|         | DynConv [43]   | ✓       | S    | 7.47        | 51.3          |
|         | FBS [5]        | ✓       | C    | 8.02        | 55.7          |
|         | BAS [7]        | ✓       | C    | 7.79        | 64.2          |
| FGC (Ours) | ✓       | C    | 7.26 | 65.0        |
| FGC (Ours) | ✓       | C    | 7.43 | 66.9        |
| ResNet-56 | Baseline | -       | -    | 5.86        | 0.0           |
|         | SFP [42]       | ×       | C    | 7.74        | 52.6          |
|         | FPGM [1]       | ×       | C    | 6.51        | 52.6          |
|         | HRank [22]     | ×       | C    | 6.83        | 50.0          |
|         | DSA [14]       | ×       | CN   | 7.09        | 52.2          |
|         | Hinge [11]     | ×       | C    | 6.31        | 50.0          |
|         | DHP [3]        | ×       | W    | 6.42        | 50.9          |
|         | ResRep [45]    | ×       | C    | 6.29        | 52.9          |
|         | RL-MCTS [10]   | ×       | C    | 6.44        | 55.0          |
|         | FBS [5]        | ✓       | C    | 6.48        | 53.6          |
|         | BAS [7]        | ✓       | C    | 6.43        | 62.9          |
| FGC (Ours) | ✓       | C    | 6.09 | 62.2        |
| FGC (Ours) | ✓       | C    | 6.41 | 66.2        |
modules.

When extending to WideResNet [36] and MobileNetV2 [37], FGC still achieves consistent improvements, as shown in Table 6. On WRN-28-10, FGC prunes 3.6% more computation while achieving 0.3% less error. FGC could even further reduce redundancy in the lightweight network. On MobileNetV2, FGC prunes 4.5% more computation and achieves 0.2% less error. The results reflect FGC’s generalizability on various network architectures.

| Model       | Method | Error (%) ↓ | Pruning (%) ↑ |
|-------------|--------|-------------|---------------|
| WRN-28-10   | Baseline | 5.13        | 0             |
|             | w/o FGC | 6.15        | 42.5          |
|             | FGC     | 5.85        | 46.1          |
| MobileNetV2 | Baseline | 5.19        | 0             |
|             | w/o FGC | 6.27        | 38.8          |
|             | FGC     | 6.07        | 43.3          |

Table 6: Comparisons of the pruned WideResNet and MobileNetV2 on CIFAR10.

| Model       | Method | Dynamic | Error (%) ↓ | Pruning (%) ↑ |
|-------------|--------|---------|-------------|---------------|
| ResNet-20   | Baseline | -      | 31.38       | 0.0           |
|             | BAS [7]  | ✓       | 31.98       | 26.2          |
|             | FGC (Ours) | ✓       | 31.37       | 26.9          |
|             | FGC (Ours) | ✓       | 31.69       | 31.9          |
| ResNet-32   | Baseline | -      | 29.85       | 0.0           |
|             | CAC [47] |   ✓     | 30.49       | 30.1          |
|             | BAS [7]  | ✓       | 30.03       | 39.7          |
|             | FGC (Ours) | ✓       | 29.90       | 40.4          |
|             | FGC (Ours) | ✓       | 30.09       | 45.1          |
| ResNet-56   | Baseline | -      | 28.44       | 0.0           |
|             | CAC [7]  |   ✓     | 28.69       | 30.0          |
|             | BAS [7]  | ✓       | 28.37       | 37.1          |
|             | FGC (Ours) | ✓       | 27.87       | 38.1          |
|             | FGC (Ours) | ✓       | 28.00       | 40.7          |

Table 7: Comparisons of the pruned ResNet on CIFAR100.

CIFAR100. In Table 7 we present the pruning results of ResNet-{20, 32, 56}. For example, after pruning 26.9% computational cost of ResNet-20, FGC
Table 8: Comparisons of the pruned ResNet on ImageNet.

| Model   | Method | Dynamic | Type | Top-1 (%) | Top-5 (%) | Pruning (%) |
|---------|--------|---------|------|-----------|-----------|-------------|
| ResNet-18 | Baseline | -       | -    | 30.15     | 10.92     | 0.0         |
|         | SFP [42] | ✗       | C    | 32.90     | 12.22     | 41.8        |
|         | FPGM [1] | ✗       | C    | 31.59     | 11.52     | 41.8        |
|         | PFP [45] | ✗       | C    | 34.35     | 13.25     | 43.1        |
|         | DSA [14] | ✓       | CN   | 31.39     | 11.65     | 40.0        |
|         | LCCN [15] | ✓    | S    | 33.67     | 13.06     | 34.6        |
|         | AIG [19] | ✓       | L    | 32.01     | -         | 22.3        |
|         | CGNet [9] | ✓      | C    | 31.70     | -         | 50.7        |
|         | FBS [5] | ✓       | C    | 31.83     | 11.78     | 49.5        |
|         | BAS [7] | ✓       | C    | 31.66     | -         | 47.1        |
|         | FGC (Ours) | ✓    | C    | 30.67     | 11.07     | 42.2        |
|         | FGC (Ours) | ✓    | C    | 31.49     | 11.62     | 48.1        |

even achieves 0.01% less classification error. For ResNet-32 and ResNet-56, FGC outperforms SOTA methods CAC [17] and BAS [7] with significant margins, from perspectives of both test error and computation reduction. Specifically, FGC, compared with BAS, reduces 5.4% and 3.6% more computation of ResNet-32 and ResNet-56 without bringing more errors. Experimental results on CIFAR100 justify the generality of our method to various datasets.

**ImageNet.** In Table 8 we further compare FGC with methods on the large-scale dataset to prune ResNet-18. One can see that FGC outperforms most static pruning methods. FPGM [1] reduces 41.8% computation with 31.59% top-1 error, while FGC reduces 42.2% computation with 30.67% top-1 error. It is comparable with other dynamic pruning methods. For example, compared with BAS [7], FGC prunes more computation (48.1% vs 47.1%) with lower top-1 error (31.49% vs 31.66%). FGC outperforms AIG [19] and LCCN [15] significantly with much higher computation reduction and lower errors. Experimental results demonstrate our method’s effectiveness in solving the complex representation redundancy on the large-scale image dataset.
4.5. Transferring Tasks

The proposed gated models can be generalized to object detection and semantic segmentation tasks.

### Table 9: Object detection on PASCAL VOC 2007 dataset with Faster R-CNN.

| Backbone     | Method  | mAP (%) | Pruning (%) |
|--------------|---------|---------|-------------|
| ResNet-34    | Baseline| 73.57   | 0           |
|              | w/o FGC | 67.66   | 50.6        |
|              | FGC     | 69.18   | 52.0        |
| ResNet-50    | Baseline| 74.77   | 0           |
|              | w/o FGC | 71.38   | 34.3        |
|              | FGC     | 71.81   | 34.3        |

**Object Detection.** Experiments are conducted on PASCAL VOC 2007 using the Faster R-CNN method. The mAP (mean average precision) and the average pruning ratio are used to evaluate the accuracy-computation trade-off. The average pruning ratio is calculated based on the backbone. As shown in Table 9, with a slight accuracy drop, FGC brings the detector with more than 50% computation reduction. Note that FGC improves the trade-off compared with vanilla gated models. It achieves 0.48% higher mAP with approximate computation of vanilla gated model on ResNet-50 backbone. And FGC achieves 1.52% higher mAP with more reduction (52.0% vs 50.6%) on ResNet-34. These experiments indicate that our method could better utilize instance-wise redundancy to derive compact sub-networks for each input candidate.

### Table 10: Semantic segmentation on Cityscapes dataset with PSPNet.

| Backbone     | Method  | mIoU (%) | PixelACC (%) | Pruning (%) |
|--------------|---------|----------|--------------|-------------|
| ResNet-50    | Baseline| 71.68    | 96.08        | 0           |
|              | BAS     | 71.90    | 93.50        | 23.7        |
|              | w/o FGC | 71.70 (+0.02) | 96.18 (+0.10) | 34.7        |
|              | FGC     | 72.32 (+0.64) | 96.18 (+0.10) | 34.6        |

**Semantic Segmentation.** We conduct semantic segmentation experiments
on Cityscapes using PSPNet. We use mean IoU (Intersection over Union), pixel accuracy, and average pruning ratio to evaluate gated models. As shown in Table 10, the baseline model achieves 71.68% IoU and 96.08% pixel accuracy. Our gated model performs better segmentation results with significant computation reduction in comparison. Furthermore, FGC achieves 72.32% IoU and 96.18% pixel accuracy with 34.6% computation reduction, outperforming the vanilla gated model, which achieves 71.70% IoU with an approximate computational cost. It is concluded that our method achieves a better accuracy-computation trade-off.

5. Conclusions

We propose a self-supervised feature-gate coupling (FGC) method to reduce the distortion of gated features by regularizing the consistency between feature and gate distributions. FGC takes the instance neighborhood relationships in the feature space as the objective while pursuing the instance distribution coupling with the gate space by aligning instance neighborhood relationships in the feature and gate spaces. FGC utilizes the $k$NN method to explore the instance neighborhood relationships in the feature space and utilizes CSL to regularize gating modules with the generated self-supervisory signals. Experiments validate that FGC improves DNP performance with striking contrast with the state-of-the-art methods. FGC provides fresh insight into the network pruning problem. When it comes to scenarios with long-tailed distribution data or incremental unseen classes, there may be difficulties in fitting the feature distributions for gate space alignment due to biased distribution estimates. Distribution calibration methods [49, 50] that leverage implicit class relationships may raise insights to alleviate the issue. We leave these interesting problems for future exploration.
Acknowledgment

This work was supported by National Natural Science Foundation of China (NSFC) under Grant 61836012, 61771447 and 62006216, the Strategic Priority Research Program of Chinese Academy of Sciences under Grant No. XDA27000000.

References

[1] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2019, pp. 4340–4349.

[2] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, R. S. Feris, Blockdrop: Dynamic inference paths in residual networks, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 8817–8826.

[3] Y. Li, S. Gu, K. Zhang, L. V. Gool, R. Timofte, DHP: differentiable meta pruning via hypernetworks, in: Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 608–624.

[4] J. Luo, J. Wu, Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference, Pattern Recognit. (PR) 107 (2020) 107461.

[5] X. Gao, Y. Zhao, L. Dudziak, R. D. Mullins, C. Xu, Dynamic channel pruning: Feature boosting and suppression, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2019.

[6] W. Hua, Y. Zhou, C. D. Sa, Z. Zhang, G. E. Suh, Channel gating neural networks, in: Proc. Adv. Neural Inform. Process. Syst. (NeurIPS), 2019, pp. 1884–1894.

[7] B. Ehteshami Bejnordi, T. Blankevoort, M. Welling, Batch-shaping for learning conditional channel gated networks, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2020.
[8] Z. Wu, Y. Xiong, S. X. Yu, D. Lin, Unsupervised feature learning via non-parametric instance discrimination, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 3733–3742.

[9] W. Chen, P. Wang, J. Cheng, Towards automatic model compression via a unified two-stage framework, Pattern Recognit. (PR) 140 (2023) 109527.

[10] Y. Chen, Z. Ma, W. Fang, X. Zheng, Z. Yu, Y. Tian, A unified framework for soft threshold pruning, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2023.

[11] Y. Li, S. Gu, C. Mayer, L. V. Gool, R. Timofte, Group sparsity: The hinge between filter pruning and decomposition for network compression, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 8015–8024.

[12] S. Guo, B. Lai, S. Yang, J. Zhao, F. Shen, Sensitivity pruner: Filter-level compression algorithm for deep neural networks, Pattern Recognit. (PR) 140 (2023) 109508.

[13] Y. Hou, Z. Ma, C. Liu, Z. Wang, C. C. Loy, Network pruning via resource reallocation, Pattern Recognit. (PR) 145 (2024) 109886.

[14] X. Ning, T. Zhao, W. Li, P. Lei, Y. Wang, H. Yang, DSA: more efficient budgeted pruning via differentiable sparsity allocation, in: Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 592–607.

[15] X. Dong, J. Huang, Y. Yang, S. Yan, More is less: A more complicated network with less inference complexity, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2017, pp. 1895–1903.

[16] Z. Xie, Z. Zhang, X. Zhu, G. Huang, S. Lin, Spatially adaptive inference with stochastic feature sampling and interpolation, in: Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 531–548.
[17] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, K. Q. Weinberger, Multi-scale dense networks for resource efficient image classification, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2018.

[18] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 770–778.

[19] A. Veit, S. J. Belongie, Convolutional networks with adaptive inference graphs, in: Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 3–18.

[20] C. Ahn, E. Kim, S. Oh, Deep elastic networks with model selection for multi-task learning, in: Proc. IEEE Int. Conf. Comput. Vis. (ICCV), 2019, pp. 6528–6537.

[21] X. Qian, F. Liu, L. Jiao, X. Zhang, X. Huang, S. Li, P. Chen, X. Liu, Knowledge transfer evolutionary search for lightweight neural architecture with dynamic inference, Pattern Recognit. (PR) 143 (2023) 109790.

[22] S. Elkerdawy, M. Elhoushi, H. Zhang, N. Ray, Fire together wire together: A dynamic pruning approach with self-supervised mask prediction, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2022, pp. 12444–12453.

[23] R. Hadsell, S. Chopra, Y. LeCun, Dimensionality reduction by learning an invariant mapping, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2006, pp. 1735–1742.

[24] J. Huang, Q. Dong, S. Gong, X. Zhu, Unsupervised deep learning by neighbourhood discovery, in: Proc. Int. Conf. Mach. Learn. (ICML), 2019, pp. 2849–2858.

[25] K. He, H. Fan, Y. Wu, S. Xie, R. B. Girshick, Momentum contrast for unsupervised visual representation learning, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 9726–9735.
[26] C. Zhuang, A. L. Zhai, D. Yamins, Local aggregation for unsupervised learning of visual embeddings, in: Proc. IEEE Int. Conf. Comput. Vis. (ICCV), 2019, pp. 6001–6011.

[27] Y. Zhang, C. Liu, Y. Zhou, W. Wang, W. Wang, Q. Ye, Progressive cluster purification for unsupervised feature learning, in: Proc. Int. Conf. Pattern Recognit. (ICPR), 2020, pp. 8476–8483.

[28] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, A. Joulin, Unsupervised learning of visual features by contrasting cluster assignments, in: Proc. Adv. Neural Inform. Process. Syst. (NeurIPS), 2020.

[29] Y. Tian, D. Krishman, P. Isola, Contrastive multiview coding, in: Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 776–794.

[30] A. van den Oord, Y. Li, O. Vinyals, Representation learning with contrastive predictive coding, arXiv preprint arXiv:1807.03748.

[31] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2017.

[32] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, 2009.

[33] J. Deng, W. Dong, R. Socher, L. Li, K. Li, F. Li, Imagenet: A large-scale hierarchical image database, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2009, pp. 248–255.

[34] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, A. Zisserman, The pascal visual object classes (voc) challenge, Int. J. Comput. Vis. (IJCV) 88 (2009) 303–338.

[35] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, The cityscapes dataset for semantic urban scene understanding, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 3213–3223.
[36] S. Zagoruyko, N. Komodakis, Wide residual networks, in: Proc. Brit. Mach. Vis. Conf. (BMVC), 2016.

[37] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, L. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 4510–4520.

[38] S. Ren, K. He, R. B. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 39 (2015) 1137–1149.

[39] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, S. J. Belongie, Feature pyramid networks for object detection, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2017, pp. 936–944.

[40] H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia, Pyramid scene parsing network, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2017, pp. 6230–6239.

[41] L. McInnes, J. Healy, UMAP: uniform manifold approximation and projection for dimension reduction, arXiv preprint arXiv:1802.03426.

[42] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: Proc. Int. Joint Conf. Artif. Intell. (IJCAI), 2018, pp. 2234–2240.

[43] T. Verelst, T. Tuytelaars, Dynamic convolutions: Exploiting spatial sparsity for faster inference, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 2317–2326.

[44] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, Hrank: Filter pruning using high-rank feature map, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 1526–1535.

[45] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, G. Ding, Resrep: Lossless cnn pruning via decoupling remembering and forgetting, in: Proc. IEEE Int. Conf. Comput. Vis. (ICCV), 2021, pp. 4490–4500.
[46] Z. Wang, C. Li, Channel pruning via lookahead search guided reinforcement learning, in: Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV), 2022, pp. 3513–3524.

[47] Z. Chen, T. Xu, C. Du, C. Liu, H. He, Dynamical channel pruning by conditional accuracy change for deep neural networks, IEEE Trans. Neural Netw. Learn. Syst. (TNNLS) 32 (2) (2021) 799–813.

[48] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, D. Rus, Provable filter pruning for efficient neural networks, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2020.

[49] S. Yang, L. Liu, M. Xu, Free lunch for few-shot learning: Distribution calibration, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2021.

[50] B. Li, C. Liu, M. Shi, X. Chen, X. Ji, Q. Ye, Proposal distribution calibration for few-shot object detection, IEEE Trans. Neural Netw. Learn. Syst. (TNNLS) (2023) 1–8.