KNOWLEDGE TRANSFER GRAPH FOR DEEP COLLABORATIVE LEARNING

A PREPRINT

Soma Minami, Tsubasa Hirakawa, Takayoshi Yamashita, Hironobu Fujiyoshi
Chubu University
1200 Matsumotocho, Kasugai, Aichi, Japan
{minami@mprg.cs, hirakawa@mprg.cs, yamashita@isc, fujiyoshi@isc}.chubu.ac.jp

ABSTRACT

We propose Deep Collaborative Learning (DCL), which is a method that incorporates Knowledge Distillation and Deep Mutual Learning, and represents graph using a more generalized knowledge transfer method. DCL is represented by a directional graph where each model is represented by a node, and the propagation of knowledge from the source node to the target node is represented by edges. In DCL, a hyperparameter search can be used to search for an optimal knowledge transfer graph. We also propose four types of gate structure to control the propagation of gradients through the network for edges. When searching a knowledge transfer graph, optimization is performed to maximize the recognition rate of optimization target node using collaborative learning network types and gate types as hyperparameters. Using the CIFAR-100 dataset to search for an optimal knowledge transfer graph structure, we obtained a graph structure learning method that combines Knowledge Distillation with Deep Mutual Learning. Also, in experiments with the CIFAR-10, CIFAR-100 and Tiny-ImageNet datasets, we achieved a significant improvement in accuracy without increasing the network parameters beyond the vanilla model. We also show that an optimized graph can be transferred to a different dataset.

1 Introduction

In ordinary society, the presence of a good teacher contributes greatly to the learning efficiency of the students. Another effective way of making learning more efficient is to have the students learn while exchanging useful information with one another. This general principle can also be applied to the training of neural networks. Generally, supervised learning takes place in a single network using only labels. However, it is known that better accuracy can be achieved if learning is performed using multiple networks that can transfer knowledge among themselves [6, 15, 21, 17, 9].

Knowledge Distillation (KD) [6] and Deep Mutual Learning (DML) [21] are two techniques that implement knowledge transfer. KD is a technique where a superior pre-trained network with many parameters is used to train a network with few parameters. The network with many parameters is called the teacher network, and the network with few parameters is called the student network. In the training phase, the teacher network is frozen and only the student network is trained. In DML, on the other hand, untrained student networks train one another without using a trained teacher network. It is known that accuracy is improved not only by combining a large network and a small network, but also by combining networks having the same structure. In addition, three or more students can be trained simultaneously. However, when performing collaborative learning with three or more networks, no method for selecting the optimal combination of network architectures has been shown, so training is performed with same architectures. The loss functions defined between networks are also identical. As a result, the information transferred between networks becomes uniform, and it is difficult to create diversity. When training many networks collaboratively, there should be multiple teacher and student networks playing different roles.

In this study, we propose Deep Collaborative Learning (DCL), which is a generalized method incorporating conventional techniques such as KD and DML in graphical representation. DCL is represented by a directional graph where each model is represented by a node, and the propagation of knowledge from the source node to the target node is represented

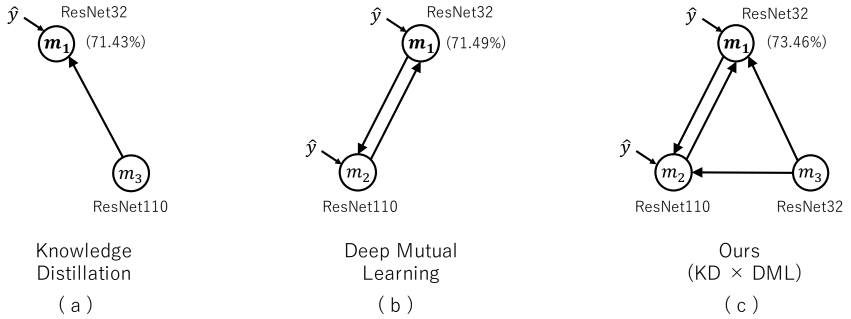


Figure 1: KD vs DML vs Ours: Knowledge transfer graphs representations optimized by (a) knowledge distillation, (b) deep mutual learning, and (c) the proposed method, which was optimized and containes both KD and DML. Each node represents a model, and each edge represents the direction of knowledge propagation. m_1 is a target node for maximizing the performance and, m_2 and m_3 are network models as teacher or student. $\hat{y}$ is a teacher label This graph representation can represent many learning methods including conventional methods such as KD and DML. In the proposed method depicted in (c), a knowledge transfer graph combining KD and DML is obtained by a hyperparameter search, resulting in greater accuracy than that of KD and DML individually.

by edges. Using this sort of representation, as shown in Fig. 1, it is possible to produce graphical representations of KD, DML, and methods that combine both of these techniques. In DCL, propagation losses are controlled by defining a separate loss function for each edge and introducing a gate structure inside the loss function. This allows diverse learning methods to be expressed by the combination of a selected model and loss function. In this study, we performed hyperparameter search with models, loss functions and gates as hyperparameters.

The contributions of this study are as follows: (1) We propose Deep Collaborative Learning (DCL), whereby collaborative learning is performed using multiple networks. In DCL, various learning methods can be expressed using a knowledge transfer graph. Hyperparameter search can be used to obtain an optimal knowledge transfer graph automatically. Searching is performed using the models defined at the nodes and the loss functions defined at the edges as hyperparameters. (2) We propose a gate structure to control the propagation of losses. We also define four types of gates with different roles: through gates, cutoff gates, linear gates and correct gates. In an experiment using the CIFAR-100 dataset, we confirmed that the use of gates to control gradient information contributes to the improvement of accuracy. (3) When we conducted a hyperparameter search using ASHA [10], we obtained a graph where KD and DML were merged together, and we discovered that this is more accurate than the conventional method. We also found that optimized DCL achieved accuracy improvements of 1.02% with CIFAR-10, 3.83% with CIFAR-100, and 2.62% with Tiny-ImageNet, without increasing the number of network parameters. (4) Finally, we show that the optimized graph can be transferred to a different dataset from the one used for optimization. In an experiment using the CIFAR-10 and CIFAR-100 datasets, we found that similar recognition rates can be achieved even with a graph that has been optimized using a different dataset.

2 Related works

This section describes typical methods used for knowledge transfer and hyperparameter search.

2.1 Knowledge transfer

A typical example of a KD method is the transfer of knowledge from the output layer [6]. The probability distribution output by a network includes an internal representation of this network. Therefore, an internal representation of the teacher network is transferred to the student network by training the student network with the output values of the teacher network as teacher labels. However, due to the softmax function, the probability values output by the network become extremely small for probability values other than Top-1. Hinton *et al.* [6] succeeded in effectively transferring the internal representation to the student by introducing a temperature parameter to the softmax function. When there is a large accuracy gap between the teacher network and the student network, students can be effectively trained by separating them with a network that is smaller than the teacher and larger than the student (a "teacher assistant") [13]. It is also known that the accuracy can be improved even when the teacher network uses the same model as the student network [4]. In addition to methods where knowledge is transferred from the output layer, there are also methods where knowledge is transferred from an intermediate layer [15, 18]. By transferring knowledge from an intermediate layer, even a deep network can convey the intermediate knowledge of a teacher. Furthermore, a method has also been proposed for transferring the internal representation acquired by the teacher network as an attention to the student network [20].

Deep Mutual Learning [21] is a method where learning is performed while transferring information between student networks. Unlike KD, it does not require a teacher network, which is a trained model. Even when using networks with identical structures, it is shown that accuracy is improved due to the effects of regularization. Further improvements in accuracy can be achieved by using the ensemble outputs of collaboratively trained networks as teachers, and by sharing the intermediate layers of these networks [17, 9].

2.2 Hyperparameter optimization

Various parameter optimization methods have been proposed for searching network architectures. Examples of architecture search methods include the Tree-structured Parzen Estimator Approach (TPE) [3] (which is based on Bayesian optimization), NAS [22] and ENAS [14] (which use reinforcement learning), DARTS [12] (which performs optimization by mapping a discrete parameter space to a continuous space), and random search [2] (which is a simpler but still powerful method). Random search optimization methods include the Successive Halving Algorithm (SHA) [7] and the Asynchronous Successive Halving Algorithm (ASHA) [10]. These methods take an approach that involves active pruning. SHA is an algorithm that repeatedly performs a process that consists of training a model for a certain amount of time using parameters proposed by random search, and then training the model while retaining the top $n\%$ of good trials. In ASHA, this algorithm is performed asynchronously and in parallel to search for high-accuracy parameters while increasing the computational efficiency. ASHA achieves accuracy that equals or surpasses that of other more complex methods including TPE, NAS, ENAS and DARTS [11]. The above approach is aimed at optimizing the network structure. Our method uses a hyperparameter search to optimize the structure of a knowledge transfer graph after fixing the network structure.

3 Proposed method

We describe the details of the knowledge transfer graph in DCL. In the proposed method, the direction of knowledge propagation between networks is represented by a directed graph, and a different loss function is defined for each edge as shown in Fig. 2. By defining different loss functions, it is possible to express various knowledge transfer methods.

3.1 Knowledge transfer graph representation

Define a directed graph where node m_i represents the ith model used in learning, and two edges are defined for each node. These edges represent the directions in which gradient information is transferred. The losses calculated from the outputs of the two models are back-propagated towards the target node. Errors are not back-propagated to the source node.

If information is all transferred to the target node from the source node throughout the entire training phase, the learning of the target node is liable to be disrupted. Therefore, we introduce a gate that controls whether or not a node propagates errors through edges. A gate is a mechanism that controls the gradient information of error back-propagation to a target node. The losses for each sample output by the loss function are weighted to determine which loss value to back-propagate. Fig. 2 shows the knowledge transfer graph representation for the case where the number of models M is 3.

3.2 Loss function

The mini-batch comprising the image of the nth sample $\boldsymbol{x}_n$ and the label $\hat{y}_n$ is represented as $\mathcal{B} = \{\boldsymbol{x}_n, \hat{y}_n\}_{n=1}^{N}$, and the batch size of mini-batch $\mathcal{B}$ is represented as $|\mathcal{B}|$. For the label $\hat{y}_n$, we use the number of the correct class. The number of models used for learning is M, and the source and target nodes are m_s and m_t, respectively.

When obtaining the difference in output probabilities between nodes, we use the Kullback–Leibler (KL) divergence $KL(\boldsymbol{p}_s(\boldsymbol{x}_n)||\boldsymbol{p}_t(\boldsymbol{x}_n))$. Here, $\boldsymbol{p}_s$ and $\boldsymbol{p}_t$ are the response values of the source and target nodes respectively, and consist of probability distributions normalized by the softmax function.

If the one-shot vector representation of the label $\hat{y}_n$ is $\boldsymbol{p}_{\hat{y}_n}$, then the error between $\boldsymbol{p}_{\hat{y}_n}$ and the output $\boldsymbol{p}_t(\boldsymbol{x}_n)$ of target node t is calculated using the cross-entropy function $H(\boldsymbol{p}_{\hat{y}_n}, \boldsymbol{p}_t(\boldsymbol{x}_n))$. $H(\boldsymbol{p}_{\hat{y}_n}, \boldsymbol{p}_t(\boldsymbol{x}_n))$ can be decomposed into the

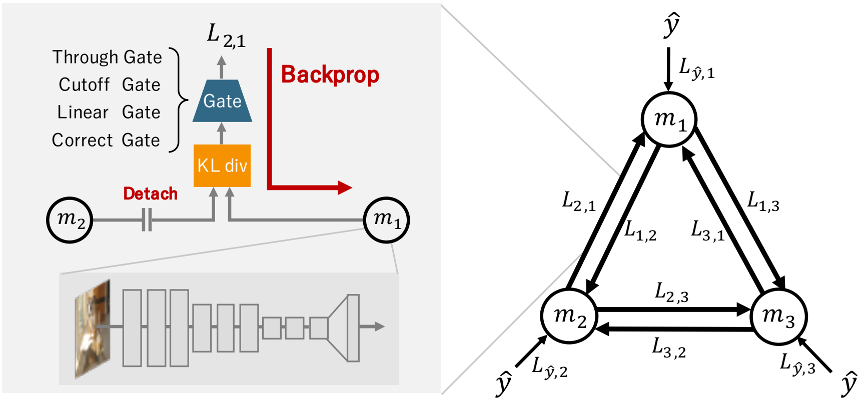


Figure 2: Knowledge transfer graph (for the 3 node case): Each node represents a model, and a loss function $L_{s,t}$ is defined for each edge. $\hat{y}$ is a label. $L_{s,t}$ calculates the KL divergence from the outputs of two nodes, and is calculated by passing it through a gate function. The calculated loss gradient information is only propagated in the direction of the arrow. We can also represent unidirectional knowledge transfer by cutting off edges with a cutoff gate.

sum of KL divergence and entropy as follows:

$$\begin{aligned} H(\boldsymbol{p}_{\hat{y}_n}, \boldsymbol{p}_t(\boldsymbol{x}_n)) &= KL(\boldsymbol{p}_{\hat{y}_n} || \boldsymbol{p}_t(\boldsymbol{x}_n)) + H(\boldsymbol{p}_{\hat{y}_n}, \boldsymbol{p}_{\hat{y}_n}) \\ &= KL(\boldsymbol{p}_{\hat{y}_n} || \boldsymbol{p}_t(\boldsymbol{x}_n)). \end{aligned} \tag{1}$$

Here, since $\boldsymbol{p}_{\hat{y}_n}$ is a one-hot vector, its entropy $H(\boldsymbol{p}_{\hat{y}_n}, \boldsymbol{p}_{\hat{y}_n})$ is zero, and so $H(\boldsymbol{p}_{\hat{y}_n}, \boldsymbol{p}_t(\boldsymbol{x}_n)) = KL(\boldsymbol{p}_{\hat{y}_n} || \boldsymbol{p}_t(\boldsymbol{x}_n))$. Therefore, the error between the label and the output can also be represented by the KL divergence in the same way as the error between the node outputs. In the following, $\boldsymbol{p}_{\hat{y}_n}$ is denoted by $\boldsymbol{p}_0(\boldsymbol{x}_n)$.

$L_{s,t}$ represents the loss function used when knowledge is propagated from the source node m_s to the target node m_t, which is defined by:

$$L_{s,t} = \sum_{n}^{|\mathcal{B}|} G_{s,t}(KL(\boldsymbol{p}_s(\boldsymbol{x}_n) || \boldsymbol{p}_t(\boldsymbol{x}_n))), \tag{2}$$

where $G_{s,t}(\cdot)$ is a gate function. The gate determines which value is to be back-propagated out of the KL divergence values calculated for each sample.

Finally, the loss function of the target node m_t is expressed as the sum of losses for all nodes as follows:

$$L_t = \sum_{s=0, s \neq t}^{M} L_{s,t}. \tag{3}$$

3.3 Gates

Gates weight the losses to control the propagation of gradients through the network. We define four types of gate: through gate, cutoff gate, linear gate and correct gate. A through gate simply passes through the losses of each input sample without any changes.

$$G_{s,t}^{Through}(a) = a \tag{4}$$

A cutoff gate is a gate that performs no loss calculation. It can be used to cut off any edge in a knowledge transfer graph. This function is required in methods such as KD where knowledge transfer is only performed in one direction.

$$G_{s,t}^{Cutoff}(a) = 0 \tag{5}$$

A linear gate changes its loss weighting linearly with time during training. It has a small weighting at the initial epoch, and its weighting becomes larger as training progresses.

$$G_{s,t}^{Linear}(a) = \frac{k}{k_{end}} a \tag{6}$$

Algorithm 1 Network Optimization

```
Input: Number of nodes $M$, number of epochs $E$
Initialize: Initialize all network weightings, or read in the weightings of a trained network
    for _ = 1 to E do
        Input the same image to each network $m_n$, and obtain the response value $\boldsymbol{p}_n$.
        Obtain the loss $L_n$ according to Eq. (3).
        Obtain the update quantity of $m_n$ from the gradient $L_n$.
        Update the weighting of all the networks.
    end for
```

Here, k is the cumulative number of updates, and k_{end} is the number of updates at the end of training.

A correct gate is a gate that only passes the losses of samples whose source node is correct. If the Top-1 class number of a source node m_s is y_s, then a correct gate can be expressed as follows:

$$G_{s,t}^{Correct}(a; \hat{y}, y_s) = \delta_{\hat{y},y_s} \cdot a. \tag{7}$$

When the source node is not a pre-trained model, the propagation of false information can be suppressed at the initial epoch. While a linear gate weights the overall loss, a correct gate selects the samples from which the loss is calculated.

3.4 Network optimization

Algorithm 1 shows how network updates are performed during training. First, all the model weights are initialized with random numbers unless all the gates $G_{i,t}$ corresponding to node m_i are cutoff gates, in which case m_i is initialized with the weighting of the pre-trained model. The trained model is trained only with the labels, using the same dataset as the one used for the hyperparameter search. Here, m_i is frozen during training and its weighting is not updated. This node performs a role equivalent to that of the teacher network used in KD.

The losses are obtained by inputting the same samples to all the nodes. Gradients are obtained from the resulting losses, and all the nodes are updated simultaneously. The gradient of loss L_t obtained from Eq. (3) is back-propagated only to node m_t, and has no effect on the other nodes. In DML, after updating the weighting of the first node, the picture is input again to the updated nodes to obtain a response value. The losses between every node are then recalculated from this response value, and gradient descent is performed for the second node. These steps are repeated until every node has been updated. However, this updating method causes a significant increase in computational cost as the number of nodes increases. In DCL, since the weighting of every node is updated during a single forward calculation, it is possible to reduce the computational cost during training.

3.5 Graph optimization

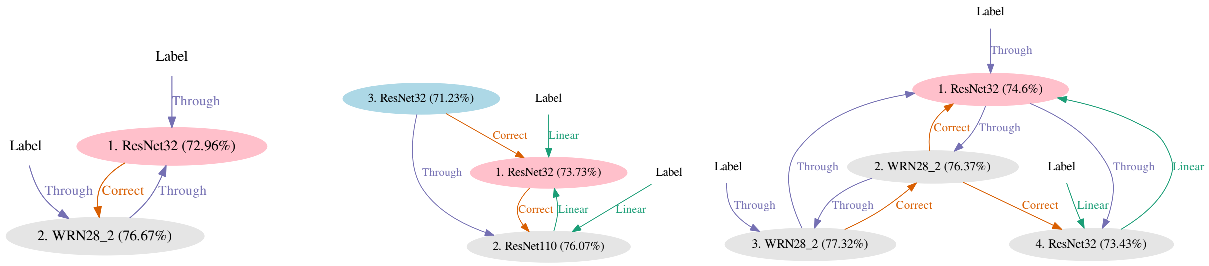
A target node to be optimized is specified, and the knowledge transfer graph is optimized to maximize the accuracy of this node. The hyperparameters to be optimized are the model type and gate type. We used the Asynchronous Successive Halving Algorithm (ASHA) [10] as the hyperparameter optimization method. First, using D computation nodes, we randomly create a knowledge transfer graph with D nodes, and perform distributed asynchronous learning. In each knowledge transfer graph, the accuracy of the optimization target node is evaluated using verification data at epochs $1, 2, 4, \cdots, 2^k$. If this accuracy is in the lower 50% of all the accuracy values evaluated in the past, then the graph is abandoned and training is performed again after generating a new graph. This process is repeated until the total number of trials reaches T. ASHA can achieve improvements in terms of both temporal efficiency and accuracy by performing a random search with active early termination in a parallel distributed environment. we performed optimization with $D = 30$ and $T = 1500$.

4 Experiments

We verified the efficacy of DCL with a knowledge transfer graph optimized by ASHA.

4.1 Experimental setting

Dataset: We used the CIFAR-10, CIFAR-100 [8] and Tiny-ImageNet [1] datasets, which are used for general object recognition. CIFAR-10 and CIFAR-100 consist of 50,000 images for training and 10,000 images for verification. Both datasets consist of images with dimensions of 32×32 pixels, and include labels for 10 classes and 100 classes,



(a) 2 nodes **(b) 3 nodes** **(c) 4 nodes**

Figure 3: Optimized knowledge transfer graph: The red node is the optimization target node, the blue node is the pre-trained source node, and "Label" represents a teacher label. At each edge, the selected gate is shown. The numbers in parentheses show the accuracy achieved in one out of five trials.

respectively. Data augmentation was performed by processing the training data images with 4-pixel padding (reflection), random cropping and random flipping. Data augmentation was not applied to the verification images. The Tiny-ImageNet dataset consists of 100,000 training images and 10,000 verification images sampled from the ImageNet [16] dataset. This dataset consists of images with dimensions of 64×64 pixels, and labels with 200 classes. The data augmentation settings were the same as for the CIFAR datasets.

Models: We used three typical network: ResNet32, ResNet110 [5], and Wide ResNet 28-2 [19]. However, when training with Tiny-ImageNet, since the images are larger in size, the stride of the initial convolution layer was set to 1. The evaluation target node was set to ResNet32. The other nodes were selected by ASHA to achieve the best recognition rate at the evaluation target node (ResNet32).

Implementation details: For the optimization algorithm, we used SGD and Nesterov momentum in all the experiments. We used an initial learning rate of 0.1, a momentum of 0.9, and a batch size of 64. In addition, when training on CIFAR, the learning rate was reduced to one tenth every 60 epochs, for a total of 200 epochs. When training on the Tiny-ImageNet, the learning rate was reduced to one tenth at the 40th, 60th and 70th epochs, for a total of 80 epochs. The reported accuracy values are averaged over five trials with a fixed graph structure implemented after obtaining the optimized graph. The standard deviation over each set of five trials is also shown. Our experiments were implemented using the Pytorch framework for deep learning, and the Optuna framework for hyperparameter searching. The computations were performed using 90 Quadro P5000 servers. Our implementation is available at `https://github.com/somaminami/DCL`.

4.2 Optimized knowledge transfer graphs

Fig. 3 shows the results obtained when knowledge graphs with 2, 3 and 4 nodes were optimized with the CIFAR-100 dataset. Also, Table 1 shows the accuracy achieved when each model was trained with labels only. For any number of nodes, the optimization target node has much better accuracy than was achieved in individual learning. The accuracy of nodes other than the optimization target was also improved. The graph for two nodes is similar to the graph for DML, where correct gates are only formed on edges from a smaller network to a larger one. To avoid disrupting the learning of WRN28-2, ResNet32 transfers information only when it has been corrected itself. The graph for three nodes is a fusion of KD and DML. KD-like learning is initially performed because the output of the linear gate at the initial stage of training is close to zero, but as training progresses, it is thought that learning is performed through a combination of KD and DML. The graph for four nodes performs learning with a combination of ResNet32 and Wide ResNet28-2, which are applied to two nodes each. The error information from the labels is not transferred to Wide ResNet28-2 at node 2, and learning is performed using only the outputs of nodes 1 and 3. Node 2 acts as an intermediary transferring information from node 3, and can be considered to perform the role of a teacher assistant [13].

Table 1: Accuracy of individual learning: Accuracy and standard deviation obtained when training a single network with labels only

	ResNet32	ResNet110	Wide ResNet 28-2
Accuracy [%]	70.71 ± 0.39	72.59 ± 0.54	74.60 ± 0.38

4.3 Comparison with other methods

Table 2 shows comparison result with conventional methods. “DCL” shows the results of the proposed method for optimized graphs with three or four nodes. “KD [6]” uses a trained ResNet110 network as a teacher, and sets the temperature parameter to $T = 2$. In “DML [21]” with using over three nodes, all student network are the same architecture. Since the proposed method adopts which model to use as a hyper parameter, it is possible to select the optimal combination of models. In “[17]” and “ONE [9]”, the intermediate layers of multiple networks are shared during training. Then, only layers that area close to the output layer are branched, and the ensemble output of the branched output layers is used as a teacher.

The model learned by the optimized knowledge transfer graph achieved better accuracy than KD, DML, and the latest method proposed more recently. By optimizing the knowledge transfer graph, DCL discovered a new learning method that combines both KD and DML as shown in Fig. 3b.

Table 2: Comparison with conventional methods: “*” denotes a pre-trained model. T is a temperature parameter. “**” denotes a value cited from the paper.

Method	Accuracy (Node 1)	Node 1	Node 2	Node 3	Node 4
KD ($T = 2$) [6]	71.43 ± 0.43	ResNet32	ResNet110*	-	-
DML [21]	71.49 ± 0.24	ResNet32	ResNet110	-	-
DML [21]	72.09 ± 0.43	ResNet32	ResNet32	ResNet32	-
DCL (Ours)	**73.46** ± 0.42	ResNet32	ResNet110	ResNet32*	-
DML [21]	72.76 ± 0.35	ResNet32	ResNet32	ResNet32	ResNet32
[17]	73.36** ± 0.26	(Multiple ResNet32 with shared intermediate layers)			
ONE [9]	73.48** ± N/A	(Multiple ResNet32 with shared intermediate layers)			
DCL (Ours)	**74.34** ± 0.32	ResNet32	WRN28-2	WRN28-2	ResNet32

4.4 Validity of gates on various datasets

Table 3 shows the accuracy of graphs optimized with CIFAR-10, CIFAR-100 and Tiny-ImageNet. For comparison, the results obtained with fixed gates transferred the KL loss defined at every edge directly to the source node, resulting in a DML-like method. The proposed method achieves higher accuracy than the comparative method in almost all conditions, and it is important to use of gates to control transferred information.

With two nodes, DCL does not differ much from the DML-like method, but as the number of nodes increases, the difference in accuracy becomes greater. The DML-like method has the same loss function defined between all the nodes, making it difficult to generate diversity even when the number of nodes is increased. In addition, since the edges between nodes are all connected, they also transfer gradient information that disrupts the training of the optimization target node. On the other hand, DCL can transfer only gradient information that contributes to training in the optimization target node because individual loss functions are defined for each edge.

Table 3: Results of optimization on various datasets: Using ResNet32 as the optimization target model. “Fixed” indicates all gates are through gates. “Optimized” indicates they have been optimized.

# of nodes	Gates	CIFAR10	CIFAR-100	Tiny-ImageNet
1	-	93.12 ± 0.27	70.71 ± 0.39	53.18 ± 0.08
2	Fixed (Through Gate)	93.25 ± 0.50	72.47 ± 0.78	**54.93** ± 0.29
	Optimized	**93.65** ± 0.14	**72.88** ± 0.41	54.69 ± 0.16
3	Fixed (Through Gate)	93.53 ± 0.24	71.88 ± 0.43	53.78 ± 0.78
	Optimized	**93.92** ± 0.20	**73.46** ± 0.42	**55.02** ± 0.31
4	Fixed (Through Gate)	93.01 ± 0.79	73.40 ± 0.39	53.92 ± 0.21
	Optimized	**93.99** ± 0.27	**74.34** ± 0.32	**55.80** ± 0.26
5	Fixed (Through Gate)	93.61 ± 0.23	73.40 ± 0.28	52.12 ± 0.30
	Optimized	**94.14** ± 0.16	**74.54** ± 0.59	**55.30** ± 0.16
6	Fixed (Through Gate)	93.84 ± 0.39	73.85 ± 0.45	49.37 ± 1.70
	Optimized	**94.17** ± 0.21	**74.22** ± 0.22	**55.16** ± 0.19
7	Fixed (Through Gate)	93.75 ± 0.27	73.53 ± 0.27	53.10 ± 0.44
	Optimized	**94.07** ± 0.14	**74.71** ± 0.23	**54.78** ± 0.36

4.5 Graph transfer to another dataset

We examined the accuracy achieved when a graph optimized for CIFAR-10 is trained with the verification data of CIFAR-100. CIFAR-10 is a dataset with ten classes consisting of vehicles and animals. On the other hand, CIFAR-100 is a more diverse dataset with a hundred classes, including plants, insects and furniture. The results are shown in Table 4. Even with the CIFAR-10 optimized graph, we achieved the same accuracy as with a graph optimized for CIFAR-100. This result demonstrates the generalization of knowledge transfer graphs, and shows that it is possible to transfer graphs between datasets.

Table 4: CIFAR-10 to CIFAR-100: Results obtained when training with CIFAR-100 using a graph optimized for CIFAR-10.

# of nodes	CIFAR-100	CIFAR-10 to CIFAR-100
2	**72.88** ± 0.41	72.47 ± 0.37
3	73.46 ± 0.42	**73.63** ± 0.18
4	**74.34** ± 0.32	73.76 ± 0.25
5	74.54 ± 0.59	**74.62** ± 0.24

5 Conclusion and future work

In this study, we have proposed a collaborative learning method that learns using a graph that expresses the transfer of knowledge between networks, and we have devised a way of searching optimal knowledge transfer graphs by means of a hyperparameter search. Graphs optimized in this way achieve higher accuracy than graphs produced by conventional methods. We also confirmed that the introduction of gates to control the transfer of information between networks allows networks to be trained effectively.

Since our proposed DCL method defines nodes as individual networks, it only transfers knowledge from the output layers of these networks. Future work includes knowledge transfer from an intermediate layer. It is also possible to perform knowledge transfer using the ensemble inference of multiple networks. Other interesting possibilities include the introduction of an encoder/decoder model, and the use of multitasking.

References

[1] Tiny ImageNet Visual Recognition Challenge. `https://tiny-imagenet.herokuapp.com/`.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2546–2554, 2011.

[4] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1607–1616, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[7] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning (ICML)*, pages 1238–1246, 2013.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[9] Xu Lan, Xiatian Zhu, and Shaogang Gong. Knowledge distillation by on-the-fly native ensemble. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7527–7537, 2018.

[10] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *arXiv preprint arXiv:1810.05934*, 2018.

[11] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[12] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[13] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393*, 2019.

[14] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning (ICML)*, pages 4095–4104, 2018.

[15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015.

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[17] Guocong Song and Wei Chai. Collaborative learning for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1837–1846, 2018.

[18] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4133–4141, 2017.

[19] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[20] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations (ICLR)*, 2017.

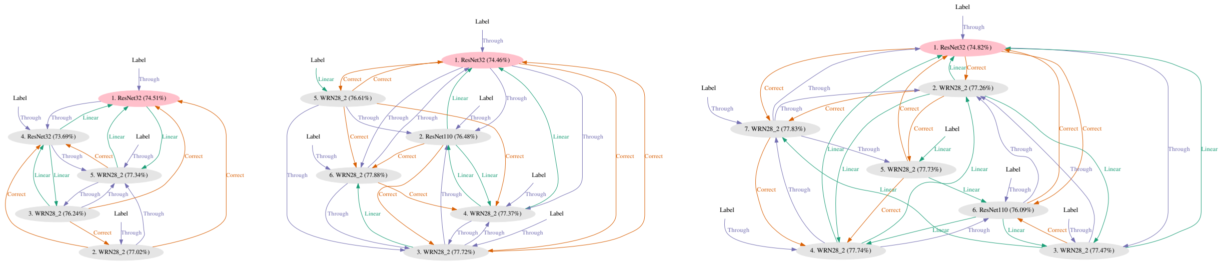
[21] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[22] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

6 Appendix

6.1 Graph visualization

Figure 4 shows visualizations of the optimized graphs with 5, 6 and 7 nodes in Sec. 4.4.



(a) 5 nodes

(b) 6 nodes

(c) 7 nodes

Figure 4: Optimized knowledge transfer graph