Don't Miss Any Piece of Knowledge: In-Network Mutual Learning with Sketch Side Branches

Anonymous ECCV submission

Paper ID 3191

Abstract. Existing knowledge distillation methods are confronted with two major challenges: (1) *High training cost* due to independent teacher and student models; (2) *Low distillation efficiency* due to the large gap between the learning capabilities of the teacher and the students, and the unidirectional teacher-to-student distillation which fails to exploit the unique knowledge learned by the students.
MUSE is a new distillation framework that addresses both the challenges at the same time. MUSE distills knowledge from multiple side branches of a single network, hence delivering low training cost. MUSE closes the learning capability gap of the branches by "sketching" the remaining network, such that each branch has a simplified yet similar structure to the full network. This similarity reduces the heterogeneity of the branches and smooths the knowledge transfer, with limited extra computational cost. MUSE then performs multilateral mutual learning among these branches, to exploit the knowledge learned by all the side branches, thus maximizing the distillation efficiency. Extensive evaluations show that MUSE not only improves end-to-end performance (by 1.5% on average on ImageNet), but also significantly enhances the side branches (by 14%), compared to state-of-the-art distillation methods. These high-quality side branches can readily serve as early exits of a network to further speed up runtime inference.

Keywords: Knowledge Distillation, Mutual Learning, Heterogeneity

1 Introduction

Recent years have witnessed a progression of convolutional neural networks (CNNs) towards being increasingly deep and large. This trend has brought both impressive performance and, at the same time, higher training and inference costs. Knowledge distillation (KD) is one of the major family of methods for training smaller networks with comparable performance. It transfers the learned knowledge from a stronger teacher model to a smaller student model, so that the student's predictions can align with the teacher, yet with lower inference cost.

A major problem of KD methods is that they usually incur *high training cost*. Due to the inherent seperation of teacher and student models, canonical KD [6, 16, 21, 22] has to first pre-train a teacher model, then uses it to teach the student. This two-phase procedure prolongs the process of acquiring a deployable model.

Online KD methods [10, 25] train the teacher and student models simultaneously, but still needs to train multiple independent models, consuming more resources.

In-network distillation is a new form of KD which is performed inside a single network to avoid the cost of training multiple networks. It introduces multiple internal side branches from a network, and then distills knowledge from the final exit (regarded as the teacher) to the branches (as the students). Although this approach reduces the training cost, existing in-network distillation method [24] exhibits two major drawbacks that severely limit the distillation efficiency.

First, the *heterogeneity of the side branches* hinders the knowledge transfer and degrades the distillation efficiency. Different side branches and the final exit of a network are naturally heterogeneous in terms of depth, width, filter, structure, etc. This heterogeneity leads to gaps of their learning capabilities and feature representation scales. We find that such gaps will result in poor distillation efficiency among the side branches (prior study [13] shows similar insight). It is therefore necessary to reduce the heterogeneity of the branches for the knowledge to flow smoothly among them.

Second, the *unidirectional exit-to-branch knowledge transfer* fails to exploit the knowledge learned by the side branches. This design follows the conventional wisdom that a teacher (the final exit) is always stronger than students (side branches), just as in traditional KD. However, we make a counter-intuitive observation that *the final exit (or a branch located deeply in the network) does not necessarily have all the knowledge of a shallow branch*. For example, in an experiment on ResNet50 with multiple exported side branches, we find that over 12% of the test samples correctly predicted by the first branch are then misclassified by at least one of the later branches (and the final exit). We should retrieve such "missed" knowledge in the network to maximize the distillation benefits.

In this paper, we propose MUSE[1], a new in-network distillation framework that addresses all of the problems above at the same time. The key idea of MUSE is to perform mutual learning among side branches with low heterogeneity and similar structures, thus to extract and share all of the knowledge in a network without being hindered by the learning capability gap.

MUSE reduces the heterogeneity of the branches through sampling layers with different depths and scales from the remaining network. This way, each side branch effectively "sketches" the full network with a simplified yet similar structure. With the sketched network structure, the shallow branches are able to acquire richer features with different scales. The branches also have exactly the same feature transformation process and composition of feature scales (just as in the full network), making it easier to transfer knowledge among them.

MUSE then applies mutual learning [25] among the sketch branches, where the branches "teach" each other to pool and share their knowledge. This mutual learning also takes care of the heterogeneity remaining in the branches (albeit minimized), by promoting the supervision from the final (strongest) exit in

[1] MUtual learning with SkEtch branches

Table 1: Properties of existing online KD methods and MUSE

	DML [25]	ONE [10]	SD [24]	MUSE
Low Training Cost	×	×	✓	✓
Sufficient Knowledge Sources	✓	×	×	✓
Low Heterogeneity	×	✓	×	✓

the early stage of training. This supervision is then gradually weakened as the shallow branches become stronger, to fully extract their knowledge.

With rich knowledge distilled from each other, the side branches in a MUSE-trained network can be readily used as early-exit classifiers to further speed up runtime inference, just as in multi-exit networks [18, 7]. Furthermore, MUSE can also improve the accuracy of state-of-the-art specially designed multi-exit network [7] with no modification to the network structure.

In summary, this paper makes the following key contributions.

- We present the Sketch Branch Construction (SBC) strategy to reduce the heterogeneity and the learning capability gap of the side branches, which enables efficient knowledge transfer among the branches.
- We propose Side-branch Mutual Learning (SML) to exploit all the knowledge learned by a network. SML is adaptive to the growth of the branches to balance the weights of the final exits and the branches in the mutual learning.
- We show that MUSE provides high-quality side branches that can be used as early-exit classifiers to speed up runtime inference. MUSE can also improve the accuracy of existing well-designed multi-exit network.
- We conduct experiments on different types of CNNs and training methods to demonstrate the high performance and generality of MUSE. Compared with prior in-network distillation method, 13.9% and 1.5% average accuracy boosts are achieved for the side branches and the final classifier on ImageNet dataset.

2 Related Work

Knowledge distillation (KD) is a model compression technique proposed by [3] and is then utilized in neural networks [6, 12, 13, 16, 21, 22]. Traditional KD tries to transfer knowledge from a pre-trained powerful teacher model to a small target student model, in forms of class posterior probabilities, feature representations, or inter-layer flow (the inner product of feature maps), etc.

Online knowledge distillation. Recent works propose online and one-stage distillation algorithms to simplify the offline and two-stage distillation training process. To highlight the difference from MUSE, we will analyze these works from the aspects of training cost, knowledge source, and the heterogeneity among the models (branches) for the distillation (summarized in Table 1).

Deep mutual learning (DML) [25] trains a pool of models simultaneously which learn collaboratively and teach each other, which enables to extract the knowledge learned by all the models. However, training multiple independent

models results in much higher time and resource cost. Moreover, DML ignores that large heterogeneity among the models would degrade the efficiency of the knowledge transfer [13].

ONE [10] copies multiple same branches from the original network, trains an ensemble of them and applies KD techniques. Copying the branches solves the heterogeneity natively, but still brings high computational cost (as each of the branches requires comparable amount of computation to the original network). Meanwhile, although the ensemble of the branches is in general a stronger network, it could lose some of the knowledge learned by certain branches.

Self-distillation (SD) [24] also proposes to distill knowledge among multiple branch classifiers within a single network. Different from ONE, SD uses a light-weight bottleneck layer [5] as side branches attached to different depths of the network, which saves the training costs of the aforementioned methods. In SD, knowledge flows unidirectionally from the final exit of the network to the internal branches, just as in canonical KD. However, we observe that the set of correct predictions of a classifier does not necessarily fall entirely into those of the deeper classifiers. For example, on a ResNet50 network trained with side branches, we find over 12% of CIFAR100 test samples predicted correctly by the first branch are *then misclassified by at least one of the later branches and the final exit*. This suggests that the internal branches also have knowledge useful for each other and for the final exit, whereas SD fails to retrieve and share such knowledge in the network. Moreover, the bottleneck structure cannot handle the inherent heterogeneity of the different branches, which further limits the distillation efficiency of SD. OFA [13] is a simultaneous work with SD which proposes an almost the same solution. A trivial difference is that OFA is directly applied on multi-exit network [7] without adding extra side branches.

Each of the above methods overcomes only one of the drawbacks, while ignores the others (Table 1). MUSE is to our knowledge the first framework that overcomes all of the three drawbacks at the same time.

Adaptive-computation-time and multi-exit network architectures. Various studies explore adaptive-computation-time (ACT) networks to achieve low-latency inference. Such networks are able to adapt their computation given certain latency budget [11, 19]. In particular, multi-exit networks [7, 18] is a new branch of ACT that emerges in recent years. Such networks allow test samples to exit the network early via side branches when they can already be inferred with high confidence. Multi-Scale DenseNet (MSDNet) [7] is a state-of-the-art multi-exit network inspired by DenseNet [8], which adopts multi-scale network architecture and dense connectivity to increase the accuracy of early exits.

We show that MUSE can train high-quality side branches that can readily serve as early exits just as in a multi-exit network. More importantly, MUSE can further improve the performance of the specially designed multi-exit networks.

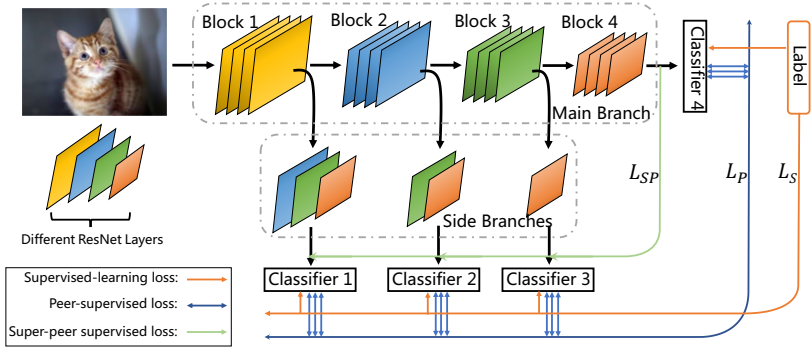


Fig. 1: An example of MUSE in a ResNet-style network equipped with multiple side branches. We attach a branch behind each layer block. The first, second and third branches consist of 3, 2 and 1 ResNet layers, respectively.

3 Method

In this section, we first introduce our side branch construction strategy to reduce the heterogeneity of the branches. Then we describe our knowledge distillation algorithm to maximize the knowledge utilization efficiency.

3.1 Sketch Branch Construction

We aim to design a side branch construction strategy that has both low branch heterogeneity and low computational cost. We leverage the opportunity that most modern CNN architectures follow a blocked structure, i.e., a flexible stack of customized blocks of layers rather tedious per-layer network piling, e.g., ResNet [5], Inception [17], NasNet [26] and many others [8, 20, 23]. Each of the blocks shares similar structure but with different weights and hyper-parameters like filter numbers, kernel sizes, etc. Features of fine scale are learned in early blocks and those of coarse scale are learned in later blocks. For simplicity and without loss of generality, in this paper we focus on such block-structured networks.

We propose *Sketch Branch Construction* (SBC) to add side branches with proper locations and structures to block-structured networks. SBC exports a side branch at the end of each block. The specific structure of a branch is determined by the depth of the block it attaches to. Specifically, we *sample one layer from each of the remaining blocks* deeper than the attaching point of the branch. For each sampled layer in a branch, we modify its hyper-parameters (e.g., filter number, stride) to reshape its output to the block it is sampled from; otherwise the input-output shapes may not match between these sampled layers. Figure 1 illustrates a concrete example where we modify a standard ResNet by adding three side branches after ResBlocks at different depths of the main branch. The

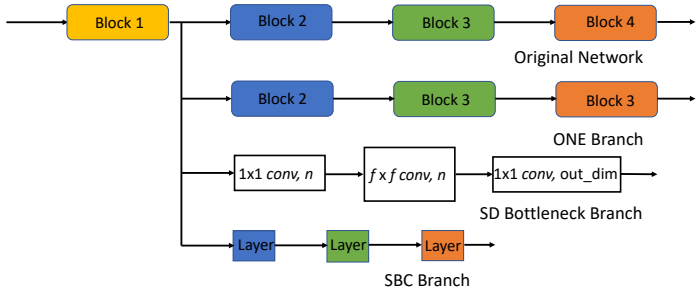


Fig. 2: Comparison of the branch structures of ONE, SD, and MUSE (SBC).

branches consist of 3, 2 and 1 ResNet layers, with each layer sampled from the deeper ResBlocks (e.g., the first layer of that block), respectively.

The intuition of this strategy is to create a "sketch" that presents a simplified yet similar structure of the remaining network. With sampled layers from different blocks, a side branch is able to acquire features with different scales within a limited branch depth. Meanwhile, the sketch ensures that features are transformed and stacked in exactly the same way as in the full network. This way, the branches not only acquire improved capacity and richer features, but also have similar feature composition to the original network. As in the example in Figure 1, the branches attached to different depths of the network all traverse layers sampled from all the remaining blocks, learning and transforming features from fine-scale to coarse-scale. SBC reduces the gap of the branches' learning capabilities, leading to higher efficiency of the mutual learning (elaborated later).

This layer sampling approach also limits the extra computational costs required for the branches. Figure 2 compares our branch structure to those of ONE and SD. ONE's branch brings obviously the most computation, because it copies the remaining network as a branch. SD uses a bottleneck block as a branch, whose computation depends on the number of hidden channels n. Our experiments show that our SBC branch has comparable or even less computation compared to SD, while providing consistently higher accuracy.

3.2 Side-Branch Mutual Learning

MUSE performs Side-branch Mutual Learning (SML) to exploit all the knowledge learned by a network. In SML, the side branches learn from each other collaboratively, in a similar manner to DML [25]. Moreover, considering the heterogeneity of the branches, SML promotes the supervison from the final exit in the early stage. It then gradually balances the weights of the final exit and the side branches, to unleash the power of mutual learning. Note that SML not only works well with the sketch branch structure described above, but is general enough to benefit other side branch structures (e.g., SD, MSDNet) as well.

We formulate the proposed side-branch mutual learning approach with a group of K ($K \geq 2$) classifiers $\{C_k\}_{k=1}^{K}$ (including corresponding branches)

attached to the same network. Given N samples $X = \{x_i\}_{i=1}^{N}$ with M classes, we denote the corresponding label set as $Y = \{y_i\}, y_i \in \{1, 2, ..., M\}$. Let z_k be the logits (the inputs fed into the softmax function of the classifier C_k.

Side-branch mutual learning defines three types of loss. The first is a conventional *supervised-learning loss* (L_S), i.e., the cross-entropy of the output of classifier C_k's softmax(z_k) and the ground-truth label y:

$$L_S(C_k) = CrossEntropy(softmax(z_k), y) \tag{1}$$

Besides the supervised-learning loss, we also borrow the *peer-supervised loss* (L_P) from traditional mutual learning [25] into the new in-network distillation setting. Specifically, we match the output of each classifier to those of all other classifiers, via a cumulative KL-divergence loss,

$$L_P(C_k) = \frac{1}{K-1} \cdot \sum_{l \neq k}^{K} KL(p_l, p_k) \tag{2}$$

where

$$p_l = softmax(z_l/\tau), p_k = softmax(z_k/\tau) \tag{3}$$

This loss effectively aligns the prediction of each classifier towards those of the other peers. Through aggregating the knowledge from different peers, this mutual learning can lead to a more robust gradient direction. The hyper-parameter τ is introduced to soften the output distribution of each classifier, thus to convey more knowledge to peers by avoiding extremely high or low probabilities.

We further introduce a *super-peer supervised loss* (L_{SP}), to consider the heterogeneous learning capabilities of classifiers. This loss induces the feature maps of all the early classifiers to fit that of a designated super-peer (we choose the deepest classifier due to the highest capability),

$$L_{SP}(C_k) = \|F_k - F_K\|_2^2 \tag{4}$$

where F_k represents the feature maps of classifier C_k (i.e., the inputs to the FC layer before softmax).

We compute the sum of the three parts of loss above, and introduce two hyper-parameters α and β controlling the tradeoff between the losses. Given K classifiers $\{C_k\}_{k=1}^{K}$, the loss function for optimizing all the classifiers is,

$$L = \sum_{k=1}^{K} ((1-\alpha)L_S(C_k) + \alpha L_P(C_k) + \beta L_{SP}(C_k)) \tag{5}$$

At the initial stage of training, the deepest classifier may learn more quickly, so we could give more bias towards this classifier to boost early learning. However, as the early classifiers' performance generally increases over time, we reduce this

bias by putting more focus on the collective learning experience. To this end, we adopt a cosine annealing policy for β:

$$\beta = 0.5 \cdot (1 + cos(\frac{epoch}{total} \cdot \pi) \cdot (\beta_{begin} - \beta_{end})) + \beta_{end} \quad (6)$$

where $\beta_{begin}, \beta_{end}$ represent the initial and final values of β. Experiments show that this policy could lead to better performance than not using, or using a constant strenghth of the super-peer supervision.

4 Experiments

We evaluate MUSE using various network architectures, knowledge distillation methods and datasets (CIFAR100 and ImageNet). Overall, our key findings are:

- MUSE improves the final-exit accuracy by 2.9% (CIFAR100) and 1.9% (ImageNet) on average compared to vanilla CNNs, and by over 1% (CIFAR100) and 1.5% (ImageNet) compared to self-distillation (SD).
- MUSE significantly improves side-branch accuracy, and outperforms SD by over 9% (CIFAR100) and 23%(ImageNet) on the first branch, with comparable computation. In particular, MUSE achieves *93%* of the final accuracy with only *18%* of the computation with ResNet152 on CIFAR100.
- MUSE also enhances existing multi-exit networks and improves the accuracy of *every* classifier (by up to 3% / 1.25% on CIFAR100 / ImageNet for MSDNet).
- MUSE effectively retrieves the unique knowledge of the side branches, which also delivers better generalization ability of the network.

We first present an end-to-end comparison between MUSE and SD. Then we conduct a ablation study to evaluate SBC and SML, respectively. We also show MUSE is applicable and benefitial to existing ACT networks. Finally, we reveal the sources of MUSE's advantages through analysis experiments.

4.1 Datasets and Setup

We conduct experiments on two popular datasets, CIFAR100 [9] and ImageNet (ILSVRC 2012) [4]. CIFAR100 contains 60,000 RGB images of 32x32 pixels with 100 classes (50,000 for training and 10,000 for test). We use random cropping, random horizontal flipping and normalization for preprocessing. ImageNet is one of the largest datasets for computer vision, with over 1.3 million images and 1000 classes. We use resizing (256x256), cropping and normalization for preprocessing.

Our models are implemented in PyTorch [14] (code available at [15]). Training details will be introduced in the corresponding experiments. All the reported accuracy numbers are Top1 accuracy. Due to limitations of computing resources, we only evaluate certain representative networks on ImageNet.

Table 2: Accuracy (%) comparison with SD on CIFAR100. "Vanilla" represents the unmodified single-exit network. Classifiers 1/4, 2/4 and 3/4 are side-branch classifiers, and 4/4 is the final exit

Network	Vanilla	Method	Classifier 1/4	Classifier 2/4	Classifier 3/4	Classifier 4/4
ResNet18	77.09	SD	67.85	74.57	78.23	78.64
		MUSE	78.93(+11.08)	79.63(+5.06)	80.13(+1.90)	80.26(+1.62)
ResNet50	77.68	SD	68.23	74.21	75.23	80.56
		MUSE	78.60(+10.37)	80.36(+6.15)	81.67(+6.44)	81.78(+1.22)
ResNet101	77.98	SD	69.45	77.29	81.17	81.23
		MUSE	78.29(+8.84)	80.47(+3.18)	82.75(+1.58)	82.54(+1.31)
ResNet152	79.21	SD	68.84	78.72	81.43	81.61
		MUSE	78.37(+9.53)	81.66(+2.94)	82.96(+1.53)	82.86(+1.25)
ResNeXt29-8	81.29	SD	71.15	79.00	81.48	81.51
		MUSE	77.68(+6.53)	79.88(+0.88)	81.62(+0.14)	81.85(+0.34)
WRN20-8	79.76	SD	68.85	78.15	80.98	80.92
		MUSE	76.80(+7.95)	78.65(+0.50)	81.01(+0.03)	81.22(+0.30)

Table 3: Accuracy (%) comparison with SD on ImageNet

Network	Vanilla	Method	Classifier 1/4	Classifier 2/4	Classifier3/4	Classifier 4/4
ResNet18	68.12	SD	41.26	51.94	62.29	69.84
		MUSE	64.39(+23.13)	66.13(+14.19)	68.64(+6.35)	70.80(+0.96)
ResNet50	76.15	SD	43.95	58.47	72.84	75.24
		MUSE	67.50(+23.55)	71.30(+12.83)	76.33(+3.49)	77.34(+2.10)

4.2 End-to-End Comparison with SD

We evaluate MUSE on classic ResNet-series networks (ResNet18/50/101/152, ResNeXt29-8 and WideResNet20-8 for CIFAR100; ResNet18/50 for ImageNet). The classifiers are located behind each of the four blocks in these networks (Figure 1). We compare MUSE against the vanilla networks (i.e., without side branches) and SD (self-distillation [24]). Unless otherwise noted, we use the results of the vanilla networks and SD reported in the SD paper.

We set the hyper-parameters $\alpha = 0.5$, $\tau = 3$ on the CIFAR100 dataset. For ImageNet we set $\alpha = 0.1$ and $\tau = 20$. For the ResNet and ResNeXt networks, we set $\beta_{begin} = 10^{-6}$ and $\beta_{end} = 10^{-7}$(except ResNet152, for which we set $\beta_{begin} = 10^{-5}$ and $\beta_{end} = 10^{-6}$, as its deeper layers lead to larger gaps between different classifiers and we use bigger β). For the WideResNet network, we set $\beta_{begin} = 10^{-5}$ and $\beta_{end} = 10^{-7}$.

As shown in Table 2, MUSE improves the vanilla network's accuracy by up to 4.6% and on average 2.9%. Compared to SD, MUSE improves the final accuracy by up to 1.6% and on average 1%. More importantly, MUSE achieves much higher side-branch accuracy than SD. For the three side-branch classifiers, MUSE's improvement is 9.05%, 3.12% and 1.94% on average, respectively. This implies that MUSE's side branches can provide high-quality predictions and have the potential of being used as early-exit classifiers (further discussed in §4.4).

We further evaluate MUSE on ImageNet. As shown in Table 3[2], our approach achieves higher accuracy than SD on all the classifiers. The improvement is over

[2] The result of vanilla ResNet50 (76.15%) is from the PyTorch official implementation [2]. The SD paper [24] reported a visibly lower result (73.56%), which we suspect was due to hyper-parameter or implementation issues.

Table 4: Accuracy (%) comparison between SBC and Bottleneck on CIFAR100

Network	Loss	Branch	Classifier 1/4	Classifier 2/4	Classifier 3/4	Classifier 4/4
ResNet18	L_S only	Bottleneck	66.54	73.11	77.63	78.19
		SBC	75.96	77.51	78.36	78.74
	SML	Bottleneck	68.18(+1.64)	74.73(+1.62)	77.86(+0.23)	78.31(+0.12)
		SBC	78.93(+1.77)	79.63(+2.12)	80.13(+1.77)	80.26(+1.52)
ResNet50	L_S only	Bottleneck	71.17	77.12	80.50	80.60
		SBC	74.86	78.29	80.40	80.35
	SML	Bottleneck	73.74(+2.57)	78.34(+1.22)	80.01(-0.49)	80.28(-0.32)
		SBC	78.60(+3.74)	80.36(+2.07)	81.67(+1.27)	81.78(+1.43)

23% and 13% on the first and second exit, and 1.53% for the final exit on average. This even bigger improvement than on CIFAR100 highlights the necessity of efficiently distilling knowledge to the shallow side branches on large datasets, where it is more difficult for them to fit the data accurately.

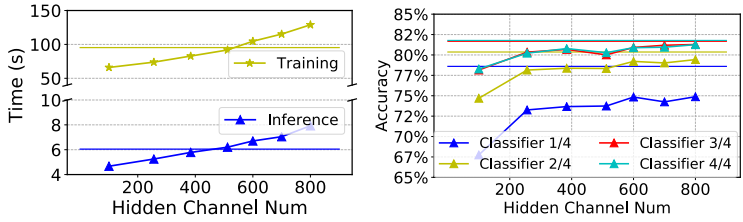
4.3 Ablation Study

Sketch Branch Construction. To evaluate the benefit of SBC, we compare the SBC branch with the Bottleneck branch of SD. We aim to reveal not only their accuracy, but also if they provide a good basis for distilling knowledge (i.e., how much gain could be obtained through knowledge distillation on these branches). To this end, for each branch we use two losses, a traditional supervised-learning loss with no knowledge distillation (i.e., L_S, Equation 1), and the SML loss (Equation 5). For both branches we use the same hyper-parameters setting.

As shown in Table 4, with the same loss, SBC achieves consistently higher accuracy than Bottleneck. Moreover, SBC always provides higher distillation improvement (i.e., the gain of SML) than Bottleneck. We even observe negative improvement with Bottleneck. This confirms the advantage that SBC improves the distillation efficiency through low-heterogeneity branches.

We also compare the computation costs of SBC against Bottleneck (Bottleneck's computation is determined by the number of hidden channels). Figure 3a shows the times required for performing training and inference for one epoch of Bottleneck branches with varying number of hidden channels (profiled on an NVIDIA 1080Ti GPU, the same below). The horizontal lines show the times of SBC. SBC's computation cost is similar to that of Bottleneck when the channel number is around 500. Note that this channel number is typically configured as 1/4 of the output channels [5], which is 512 in our case. Therefore, SBC generally requires comparable computation to Bottleneck. Moreover, as shown in Figure 3b, SBC always delivers higher accuracy than Bottleneck on every classifier, *even when Bottleneck uses more computation.* The results confirm that SBC is more efficient than Bottleneck in terms of both performance and computation.

Side-branch Mutual Learning. We compare SML against the loss of SD, which distills knowledge only from the final exit to the side branches. To focus on the effect of the loss, we replace the Bottleneck branch in SD with the same SBC branch in our approach. As shown in Table 5, SML achieves consistently higher accuracy on *every* classifier. For the final classifier, SML improves accuracy



(a) Computational Costs (b) Accuracy

Fig. 3: Computational costs (training and inference times) and accuracy of Bottleneck branches with different number of hidden channels using SML loss. The corresponding colored horizontal lines represent SBC branch (CIFAR100).

Table 5: Accuracy (%) comparison with SD plus SBC branches on CIFAR100

Network	Vanilla	Method	Classifier 1/4	Classifier 2/4	Classifier 3/4	Classifier 4/4
ResNet50	77.68	SD(SBC)	77.17	79.99	80.49	80.14
		SML	78.6(+1.43)	80.36(+0.37)	81.67(+1.18)	81.78(+1.64)
ResNet101	77.98	SD(SBC)	77.02	80.31	81.22	81.14
		SML	78.29(+1.27)	80.47(+0.16)	82.75(+1.53)	82.54(+1.40)
ResNet152	79.21	SD(SBC)	75.94	80.70	80.37	80.30
		SML	78.37(+2.43)	81.66(+0.96)	82.96(+2.59)	82.86(+2.56)
WRN20-8	79.76	SD(SBC)	74.48	77.64	79.77	79.39
		SML	76.80(+2.32)	78.65(+1.01)	81.01(+1.24)	81.22(+1.83)
WRN44-8	79.93	SD(SBC)	74.40	77.90	80.03	79.49
		SML	77.11(+2.71)	79.95(+2.05)	82.17(+2.14)	82.28(+2.79)

by over 2%. For the side-branch classifiers, SML also achieves gains by 2.03%, 0.91% and 1.74%, respectively. This is because the mutual learning utilizes more knowledge than the unidirectional knowledge transfer.

Super-peer supervised loss and the cosine anneal. We introduce an early-stage supervision from the final exit, and use a cosine-annealed hyper-parameter β to weaken this supervision as the performance of the exits increases. To demonstrate the effect of this loss, we train the networks on CIFAR100 with $\beta = 0$ (i.e., without loss), and a constant $\beta = \beta_{begin}$, then observe the difference from the cosine-annealed β used in SML. As shown in Table 6, the cosine-annealed β always achieves the highest accuracy. It brings accuracy improvement by up to 1.25% compared to $\beta = 0$, and up to 1.3% compared to the constant β. This suggests that neither the super-peer supervision nor the mutual learning is optimal if used from beginning to end. Instead, we should combine them and gradually balance their "weights" as the training proceeds to maximize the benefits of both of them.

4.4 Relative Computation and Accuracy

The previous results show the high performance of MUSE's side branches, which imply that the branches can be used for early exits in inference. Figure 4 shows the relative computation and accuracy of the three internal exits compared to those of Classifier 4/4 of three ResNet networks (on CIFAR100). We use both the FLOPs and profiled inference latency to measure the computation

Table 6: Accuracy (%) comparison with no super-peer supervised loss ($\beta = 0$) and constant β (CIFAR100)

Network	β	Classifier 1/4	Classifier 2/4	Classifier 3/4	Classifier 4/4
ResNet101	0	77.77	80.14	82.04	81.87
	10^{-6}	77.96	80.47	82.26	82.16
	annealing	**78.29**	**80.47**	**82.75**	**82.54**
ResNet152	0	77.12	80.92	82.72	82.51
	10^{-5}	77.72	80.37	82.42	82.37

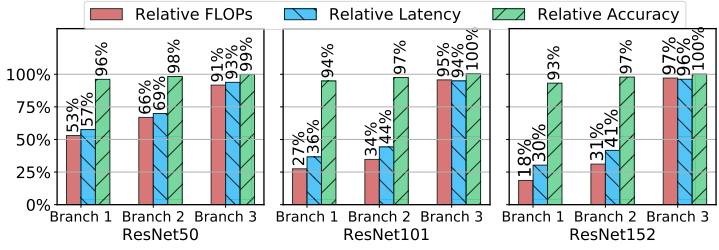


Fig. 4: FLOPs and accuracy of the three internal classifiers normalized by those of Classifier 4/4 (CIFAR100).

(latency to some extent depends on the concrete hardware configurations and software optimizations, so we also report the absolute FLOPs). For the first / second exits, the three networks all have achieved higher than 93% / 97% relative accuracy. While the required computations, especially in the two deeper networks, are pretty small. Notably, the FLOPs are only 18% / 31% (latencies 30% / 41%) in ResNet152! This significant computation savings in the deeper networks are because the block divisions are more imbalanced. The results suggest MUSE enables low-latency and resource-efficient inference without comprising performance through the side-branch classifiers.

4.5 Comparison with DML

We also compare our approach with deep mutual learning (DML [25]), which also uses the mutual learning technique, but needs to train multiple independent networks and without considerations of heterogeneity. As shown in Table 7, WRN28-10 could achieve 80.28% accuracy with DML, whereas our approach achieves nearly 82%. Moreover, our 3/4 classifier accuracy has already been higher than DML. This proves that our approach is more efficient than DML, in terms of both accuracy and resource (we only need to train a single network).

4.6 Improving Multi-Exit Networks

We demonstrate the ability of MUSE to enhance multi-exit networks by applying it on a state-of-the-art network, Multi-Scale DenseNet (MSDNet) [7]. MSDNet is specially designed for enabling multiple internal classifiers in a convolutional network. It introduces a multi-scale architecture and dense connectivity to improve

Table 7: Accuracy (%) comparison of WRN22-10 with DML on CIFAR100

Method	Accuracy			
MUSE	Classifier 1/4	Classifier 2/4	Classifier 3/4	Classifier 4/4
	77.34	79.25	81.51(+1.23)	81.95(+1.67)
DML	80.28			

Table 8: Accuracy (%) comparison of each classifier with MSDNet. Details of the four network architectures can be found in their paper [7].

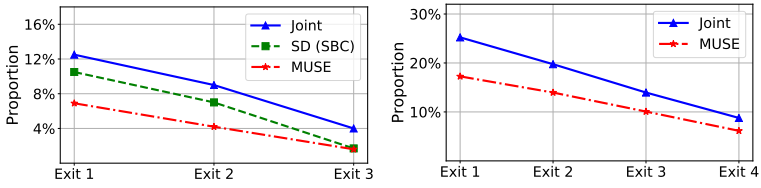
Networks	Method	Classifier 1/5	Classifier 2/5	Classifier 3/5	Classifier 4/5	Classifier 5/5
MSDNet-1	Vanilla	62.40	65.37	69.74	71.88	73.63
	MUSE	64.13(+1.73)	67.86(+2.49)	71.35(+1.61)	73.65(+1.76)	74.93(+1.30)
MSDNet-2	Vanilla	64.44	69.28	71.88	73.38	74.75
	MUSE	66.63(+2.19)	70.79(+1.51)	73.30(+1.42)	73.99(+0.61)	75.09(+0.34)
MSDNet-3	Vanilla	64.04	70.67	73.12	74.38	74.86
	MUSE	67.03(+2.99)	72.59(+1.92)	74.66(+1.54)	75.58(+1.20)	76.08(+1.22)
MSDNet-4	Vanilla	56.63	65.14	68.42	69.77	71.34
	MUSE	57.88(+1.25)	65.85(+0.71)	69.42(+1.00)	70.58(+0.81)	72.24(+0.90)

the classifiers' accuracy. In our experiments, we only modified its loss function to the SML loss defined in Equation 5, and kept the network architecture and all other configurations (e.g., learning rate, number of training epochs) exactly the same as in the PyTorch implementation released by the authors [1]. We used four network architectures for MSDNet (MSDNet-1, MSDNet-2 and MSDNet-3 for CIFAR100, MSDNet-4 for ImageNet in the results).

Table 8 shows the accuracy of each classifier on CIFAR100 and ImageNet, comparing the original MSDNet against the enhanced version by SML. SML improves the accuracy for *every* classifier, by 1.6% on average and up to 3% on CIFAR100. Similarly, on ImageNet, we achieve 0.93% accuracy improvement on average and up to 1.25%. This proves that the idea of mutual learning among the internal classifiers can also benefit well-designed multi-exit networks.

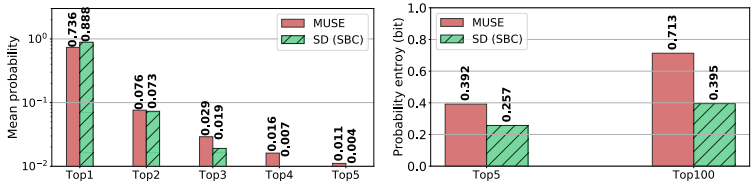
4.7 Analysis

Shallow branches also provide knowledge. Behind the idea of mutual learning among side branches is a counter-intuitive observation: shallow branches (small networks) can also "teach" deep branches (large networks). To demonstrate this, we measure for each classifier in a network the proportion of images misclassified by at least one of the later classifiers among those *correctly* classified at the current exit. Such images imply the unique knowledge of the early classifier that the later ones do not have. We train a ResNet50 network by naïve joint-training (i.e., with only the L_S defined in Equation 1), SD (SBC), and MUSE, respectively. We also conduct this measurement on MSDNet-1, comparing MUSE with the original version. As shown in Figure 5, there are always such "go-wrong-when-going-deeper" images in the models, e.g., over 25% at the first exit for the original MSDNet-1, and over 10% for ResNet50 (both naïve joint-training and SD). This implies that early classifiers also have knowledge that should be taught to the late ones. By contrast, MUSE reduces such misclassifications by transferring knowledge among all the classifiers.



(a) ResNet50 (b) MSDNet-1

Fig. 5: The proportion of images misclassified by one of the later classifiers in those correctly classified at each classifier.



(a) Mean Probability (b) Probability Entropy

Fig. 6: Generalization comparison between MUSE and SD (SBC).

Mutual learning leads to better generalization. We notice that the training accuracy is nearly 100% under both SD (SBC) and MUSE on CIFAR100 dataset, but MUSE beats SD (SBC) on test accuracy. We conjecture MUSE suppresses overfitting and leads to better generalization. To support this, we measure the mean and entropy of the network softmax probability based on ResNet50, as shown in Figure 6. We could observe (1) the mean Top1 probability is lower than SD while the other probabilities are higher; (2) the entropies of Top5 and Top100 are both higher than SD. This implies that MUSE finds a wider minima and can lead to better generalization. Compared to SD, *all* classifiers in MUSE are teachers. As the saying goes, two heads are better than one. When more teachers are teaching, the probability that all teachers are wrong is lower.

5 Conclusion

MUSE is our attempt to unleash the full power of knowledge distillation by making the best of every piece of knowledge learned by a network. We accomplish this by (1) facilitating knowledge transfer within a network through side branches, (2) smoothing the knowledge transfer through branches with similar structures and reduced heterogeneity, (3) extracting all the knowledge of a network through mutual learning, and (4) incorporating and balancing the main branch's supervision and the side branches' growth. Combined, MUSE shows great advantages in terms of performance, efficiency, and generality. Therefore, we believe MUSE will make it easier to develop and deploy more models resource-efficiently with high performance in the future.

References

1. Msdnet pytorch code. https://github.com/kalviny/MSDNet-PyTorch (2019)
2. Torchvision models. https://pytorch.org/docs/stable/torchvision/models.html (2019)
3. Buciluǎ, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 535–541. ACM (2006)
4. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
6. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
7. Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., Weinberger, K.Q.: Multi-scale dense convolutional networks for efficient prediction. arXiv preprint arXiv:1703.09844 **2** (2017)
8. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
9. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
10. Lan, X., Zhu, X., Gong, S.: Knowledge distillation by on-the-fly native ensemble. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 7528–7538. Curran Associates Inc. (2018)
11. Larsson, G., Maire, M., Shakhnarovich, G.: Fractalnet: Ultra-deep neural networks without residuals. arXiv preprint arXiv:1605.07648 (2016)
12. Lopes, R.G., Fenu, S., Starner, T.: Data-free knowledge distillation for deep neural networks. arXiv preprint arXiv:1710.07535 (2017)
13. Mirzadeh, S.I., Farajtabar, M., Li, A., Ghasemzadeh, H.: Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. arXiv preprint arXiv:1902.03393 (2019)
14. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
15. for double-blind review, A.: Self-mutual learning code (2019)
16. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: Hints for thin deep nets. arXiv preprint arXiv:1412.6550 (2014)
17. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
18. Teerapittayanon, S., McDanel, B., Kung, H.T.: Branchynet: Fast inference via early exiting from deep neural networks. In: 2016 23rd International Conference on Pattern Recognition (ICPR). pp. 2464–2469. IEEE (2016)
19. Wang, X., Yu, F., Dou, Z.Y., Darrell, T., Gonzalez, J.E.: Skipnet: Learning dynamic routing in convolutional networks. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 409–424 (2018)
20. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1492–1500 (2017)

21. Yim, J., Joo, D., Bae, J., Kim, J.: A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4133–4141 (2017)
22. Zagoruyko, S., Komodakis, N.: Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. arXiv preprint arXiv:1612.03928 (2016)
23. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)
24. Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., Ma, K.: Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In: The IEEE International Conference on Computer Vision (ICCV) (October 2019)
25. Zhang, Y., Xiang, T., Hospedales, T.M., Lu, H.: Deep mutual learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4320–4328 (2018)
26. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)