Network Transplanting

Quanshi Zhang\textsuperscript{a}, Yu Yang\textsuperscript{b}, Qian Yu\textsuperscript{c}, Ying Nian Wu\textsuperscript{b}

\textsuperscript{a}Shanghai Jiao Tong University
\textsuperscript{b}University of California, Los Angeles
\textsuperscript{c}University of California, Berkeley

Abstract

This paper focuses on a new task, i.e., transplanting a category-and-task-specific neural network to a generic, modular network without strong supervision. We design a functionally interpretable structure for the generic network. Like building LEGO blocks, we teach the generic network a new category by directly transplanting the module corresponding to the category from a pre-trained network with a few or even without sample annotations. Our method incrementally adds new categories to the generic network but does not affect representations of existing categories. In this way, our method breaks the typical bottleneck of learning a net for massive tasks and categories, i.e., the requirement of collecting samples for all tasks and categories at the same time before the learning begins. Thus, we use a new distillation algorithm, namely back-distillation, to overcome specific challenges of network transplanting. Our method without training samples even outperformed the baseline with 100 training samples.

1. Introduction

Besides end-to-end learning a black-box neural network, in this paper, we propose a new deep-learning methodology, i.e., network transplanting. Instead of learning from scratch, network transplanting aims to merge several convolutional networks that are pre-trained for different categories and tasks to build a generic, distributed neural network.

Network transplanting is of special values in both theory and practice. We briefly introduce key deep-learning problems that network transplanting deals with as follows.

1.1. Future potential of learning a universal net

Instead of learning different networks for different applications, building a universal net with a compact structure for various categories and tasks is one of ultimate objectives of AI. In spite of the gap between current algorithms and the target of learning a huge universal net, it is still meaningful for scientific explorations along this direction. Here, we list key issues of learning a universal net, which are not commonly discussed in the current literature of deep learning.

- **The start-up cost w.r.t. sample collection** is also important, besides the total number of training annotations. Traditional methods usually require people to simultaneously prepare training samples for all pairs of categories and tasks before the learning begins. However, it is usually unaffordable, when there is a large number of categories and tasks. In comparison, our method enables a neural network to sequentially absorb network modules of different categories one-by-one, so the algorithm can start without all data.
- **Massive distributed learning & weak centralized learning**: Distributing the massive computation of learning the network into local computation centers all over the world is of great practical values. There exist numerous networks locally pre-trained for specific tasks and categories in the world. Centralized network transplanting physically merges these networks into a compact universal net with a few or even without any training samples.
- **Delivering models or data**: Our delivering pre-trained networks to the computation center is usually much cheaper than collecting and sending raw training data in practice.
- **Middle-to-end semantic manipulation for application**: How to efficiently organize and use the knowledge in the net is also a crucial problem. We use different modules in the network to encode knowledge of different categories and that of different tasks. Like building LEGO blocks, people can manually connect a category module and a task module to accomplish a certain application (see Fig. 1(left)).

1.2. Task of network transplanting

To solve above issues, we propose network transplanting, i.e., building a generic model by gradually absorbing networks locally pre-trained for specific categories and tasks. We design an interpretable modular structure for a target network, namely a transplant net, where each module is functionally meaningful. As shown in Fig. 1(left), the transplant net consists of three types of modules, i.e., category modules, task modules, and adapters. Each category module extracts general features for a specific category...
Figure 1: Building a transplant net. We propose a theoretical solution to incrementally merging category modules from teacher nets into a transplant (student) net with a few or without sample annotations. The transplant net has an interpretable, modular structure. A category module, e.g., a cat module, provides cat features to different tasks. A task module, e.g., a segmentation module, serves for various categories. We show two typical operations to learn transplant nets. Blue ellipses show modules in teacher nets used for transplant. Red ellipses indicate new modules added to the transplant net. Unrelated adapters in each step are omitted for clarity.

Table 1. Comparison between network transplanting and other studies. Note that this table can only summarize mainstreams in different research directions considering the huge research diversity. Please see Section 2 for detailed discussions of related work.

| Method                          | Annotation cost | Sample Preparation | Interpretabiltiy | Catastrophic Forgetting | Modular Manipulate | Optimization  |
|---------------------------------|-----------------|--------------------|------------------|-------------------------|--------------------|---------------|
| Directly learning a multi-task net | Massive         | Simultaneously prepare samples for all tasks and categories | Low              | –                       | Not support       | back prop.    |
| Transfer- / meta- / continual-learning | Some support weakly-supervised learning | Some learn a category/task after another | Usually low | Most algorithmically alleviate | No support       | back prop.    |
| Transplanting                    | A few or no annotations | Learns a category after another | High            | Physically avoid        | Support           | back-back prop. |

We can learn an initial transplant net with very few tasks and categories in the scenario of traditional multi-task/category learning. Then, we gradually grow the transplant net to deal with more categories and tasks via network transplanting. Network transplanting can be conducted with or without human annotations as additional supervision. We summarize two typical types of transplanting operations in Fig. 1(right). The core technique is to learn an adapter to connect a task module in the transplant net and a category module from another network.

The elementary transplanting operation is shown in Fig. 2(left). We are given a transplant net with a task module \( g_S \) that is learned to accomplish a certain task for many categories, except for the category \( c \). We hope the task module \( g_S \) to deal with the new category \( c \), so we need another network (namely, a teacher net) with a category module \( f \) and a task module \( g_T \). The teacher net is pre-trained for the same task on the category \( c \). We may (or may not) have a few training annotations of category \( c \) for the task. Our goal is to transplant the category module \( f \) in the teacher net to the transplant net.

Note that we just learn a small adapter module to connect \( f \) to \( g_S \). We do not fine-tune \( f \) and \( g_S \) during the transplanting process to avoid damaging their generality.

However, learning adapters but fixing parameters of category and task modules proposes specific challenges to deep-learning algorithms. Therefore, in this study, we proposed a new algorithm, namely back distillation, to overcome these challenges. The back-distillation algorithm uses the cascaded modules of the adapter and \( g_S \) to mimic upper layers of the pre-trained teacher net. This algorithm requires the transplant net to have similar gradients/Jacobian with the teacher net w.r.t. \( f \)’s output features for distillation. In experiments, our back-distillation method without any training samples even outperformed the baseline with 100 training samples (see Table 2(left)).

1.2.1 Difference to previous knowledge transferring

Although most transfer-learning algorithms cannot be directly used to solve core problems mentioned in Section 1.1, the proposed network transplanting is close to the spirit of continual learning (or lifelong learning) [16, 6, 19, 27, 23]. As an exploratory research, we summarize our essential differences from traditional studies in Table 1.

- **Modular interpretability \( \rightarrow \) more controllability:** Besides the discrimination power, the interpretability is another important property of a neural network, which has received increasing attention in recent years. In particular, traditional transfer learning and knowledge distillation that are implemented in a black-box manner [16, 5], so the generalization process requires careful control. Whereas, our transplant net clarifies the functional meaning of each intermediate network module, which makes the knowledge-transferring process more controllable. *I.e.* the interpretable
structure clearly points out network modules that are related to the target application.

- **Bottleneck of transferring upper modules**: Most deep-learning strategies are not suitable to directly transfer pre-trained upper modules, including (i) directly optimizing the task loss on the top of the network and (ii) traditional distillation methods [11]. These methods mainly transfer low-layer features and learn new upper modules to reuse these features, rather than directly transfer pre-trained upper modules. In contrast, network transplanting just allows us to modify the lower adapter, when we transfer a pre-trained upper task module to a new category. It is not permitted to modify the upper module. This requirement physically avoids the catastrophic forgetting, but it is difficult to optimize a lower adapter if the upper is fixed (see Section 3.1 for theoretical analysis). Meanwhile, it is difficult to distill knowledge from the teacher net to the adapter. It is because except for the final network output, high-layer features in and those in the teacher net are not semantically aligned.

Thus, in this paper, the proposed back distillation first breaks the bottleneck of transferring upper modules.

- **Catastrophic forgetting**: Continually learning new jobs without hurting the performance of old jobs is a key issue for continual learning [16, 19]. Our method exclusively learns the adapter to physically prevent the learning of new categories from changing existing modules. Furthermore, when a transplant net has been constructed, we can optionally fine-tune a task/category module based on different categories/tasks to ensure the network generality.

### 1.2.2 Summarization

We can summarize contributions of this study as follows. (i) We propose a new deep-learning method, network transplanting with a few or even without additional training annotations, which can be considered as a theoretical solution to three issues in Section 1.1. (ii) We develop an optimization algorithm, i.e., back-distillation, to overcome specific challenges of network transplanting. (iii) Preliminary experiments proved the effectiveness of our method. Our method significantly outperformed baselines.

## 2. Related work

Because network transplanting is a new concept in machine learning, we would like to discuss its connections to different state-of-the-art algorithms. Firstly, we propose a new modular structure for networks, which disentangles a black-box network into different meaningful modules. Similarly, some studies have explored new representation structures instead of neural networks, such as forests and decision trees [12, 32, 7, 25] and automatic learning of optimal network structures [33, 14, 34, 31]. [20] learned a large modular neural network with thousands of sub-networks.

**Interpretability**: Unlike above studies, our dividing a network into functionally meaningful modules makes the structure interpretable. Other studies of enhancing network interpretability mainly either learn disentangled features filters/capsules in middle layers [30, 26, 17] or learn meaningful input codes of generative nets [3, 10].

**Meta-learning & transfer learning**: Meta-learning [4, 1, 13] aims to extract generic knowledge shared by different tasks/categories/models to guide the learning of a specific model. Transfer-learning methods transfer network knowledge through categories [28] or datasets [8]. Especially, continual learning [16, 6, 19, 27, 23] transfers knowledge from previous tasks to guide the new task. [16, 27] expanded the network structure during the learning process. In contrast, our study defines modular network structures with strict semantics, which allows people to semantically control the knowledge to transfer. Meanwhile, network transplanting physically avoids the catastrophic forgetting. In addition, our back distillation method solves the challenge of transferring upper modules, which is different from traditional transferring of low-layer features.

**Distillation**: [11] proposed the knowledge distillation to transfer knowledge between networks. Some recent stud-
ies [7, 25] distilled network knowledge into decision trees. [2] proposed an online distillation method to efficiently learn distributed networks. [29, 22] distilled the attention distribution/Jacobian from the teacher network to the student network, which is related to our back-distillation technique. However, these Jacobian distillation methods are hampered considering specific challenges of network transplanting (see Section 3.2 and the appendix for details). To overcome these challenges, in this study, we design a new method of back-distillation, which uses pseudo-gradients instead of using real gradients for distillation. We also balance magnitudes of neural activations between two networks, which is necessary for network transplanting.

3. Algorithm of network transplanting

Overview: As shown in Fig. 2(left), we are given a teacher net for a single or multiple tasks w.r.t. a certain category c. Let the category module f in the bottom of the teacher net have m layers, and it connects a specific task module gT in upper layers. We are also given a transplant net with a generic task module gs, which has been learned for multiple categories except for the category c.

The initial transplant net with a task module gs (before transplanting) can be learned via traditional scenario of learning from samples of some categories. We can roughly regard gs to encode generic representations for the task. Similarly, the category module f extracts generic features for multiple tasks. Thus, we do not fine-tune gs or f to avoid decreasing their generality.

Our goal is to transplant f to gs by learning an adapter h with parameters θh, so that the task module gs can deal with the new category module f.

The basic idea of network transplanting is that we use the cascaded modules of h and gs to mimic the specific task module gT in the teacher net. We call the transplant net a student net. Let x denote the output feature of the category module f given an image I, i.e. x = f(I). yT and yS are given as outputs of the teacher net and the student net, respectively. Thus, network transplanting can be described as

\[ y_T = g_T(x), \quad y_S = g_S(h(x)), \quad y_T \approx y_S \Rightarrow g_S(h(\cdot)) \approx g_T(\cdot) \]  

3.1. Problem of space projection & back-distillation

It is a challenge to let an adapter h project the output feature space of f properly to the input feature space of gs. The information bottleneck theory [24, 18] shows that a network selectively forgets certain space of middle-layer features and gradually focuses on discriminative features during the learning process (see Fig. 3(left)). Thus, both the output of f and the input of gs have vast forgotten space. Features in the forgotten input space of gs cannot pass most feature information through ReLU layers in gs and reach ys. The forgotten output space of f is referred to the space that does not contains f’s output features.

Vast forgotten feature spaces significantly boost difficulties of learning. Since valid input features of gs usually lie in low-dimensional manifolds, most features of the adapter fall inside the forgotten space. I.e. gs will not pass most information of input features to network outputs. Consequently, the adapter will not receive informative gradients of the loss for learning.

Fig. 3(right) illustrates ideal and problematic projections. A typical problematic projection is to project a feature x to a forgotten input space of gs. Another typical problem is many-to-one projections, which limit the diversity of features and decrease the representation capability of the student net. More crucially, initial many-to-one projections significantly affect the further learning process, because the back-propagation takes current space projections as anchors to fine-tune the network.

To learn good projections, we propose to force the gradient (also known as attention, Jacobian) of the student net to approximate that of the teacher, which is a necessary condition of \( g_S(h(\cdot)) \approx g_T(\cdot) \):

\[ g_S(h(\cdot)) \approx g_T(\cdot) \Rightarrow \forall J(\cdot), \quad \frac{\partial J(y_S)}{\partial x} \approx \frac{\partial J(y_T)}{\partial x} \]  

(2)

where J(·) is an arbitrary function of y that outputs a scalar. θh denote parameters of the adapter h. Therefore, we use the following distillation loss for back-distillation:

\[ \min_{\theta_h} \text{Loss} = \mathcal{L}(y_S, y^*) + \lambda \cdot \| \alpha \frac{\partial J(y_S)}{\partial x} - \frac{\partial J(y_T)}{\partial x} \|^2 \]  

(3)

where \( \mathcal{L}(y_S, y^*) \) is the task loss of the student net; \( y^* \) denotes the ground-truth label; \( \alpha \) is a scaling scalar. This formulation is similar to the Jacobian distillation in [22]. We
omitting $\mathcal{L}(y_S, y^*)$, if we learn the adapter without additional training labels.

### 3.2. Learning via back distillation

It is difficult for most recent techniques, including those for Jacobian distillation\(^1\), to directly optimize the above back-distillation loss. We briefly analyze the difficulties as follows. The minimization of the distillation loss is actually to push gradients of the student net towards those of the teacher net. To simplify the notation, we use $D_S = \frac{\partial J}{\partial x}$ and $D_T = \frac{\partial J}{\partial x}$ to denote gradients w.r.t. the feature map in the student net and the teacher net, respectively. As shown in Eqn. (4a), the computation of $D_S$ (or $D_T$) is sensitive to feature maps of layers in the $g_S$ and $h$ (those in $g_T$). Thus, it requires the student network to yield well-optimized feature maps to enable an effective distillation process. However, it is a chicken-and-egg problem—well-optimized feature maps to enable an effective distillation process. However, it is a chicken-and-egg problem—well-optimized feature maps to enable an effective distillation process. However, it is a chicken-and-egg problem—well-optimized feature maps to enable an effective distillation process. However, it is a chicken-and-egg problem—well-optimized feature maps to enable an effective distillation process.

**Computation of $G^y$:** According to Eqn. (2), $J$ can be any arbitrary function. Thus, we can enumerate functions of $J$ by randomizing different values of $G^y$. We use each $G^y$ to produce a pair of $D_S$ and $D_T$ for back distillation. For the task of object segmentation, the output is a tensor $y_S \in \mathbb{R}^{H \times W \times C}$, where $H$ and $W$ denote the height and width of the output image, and $C$ indicates the number of segmentation labels. For each image, we randomly sample $G^y \in \mathbb{R}^{H \times W \times C}$. For the task of single-category classification, the output $y_S$ is a scalar. Nevertheless, we can still generate a random matrix $G^y \in \mathbb{R}^{S \times S \times 1}$ for each image ($S = 7$ in experiments), which produces two enlarged pseudo-gradient maps $D'_S, D'_T$ for back distillation\(^1\). We normalize $G^y$ to the range of $-1 \leq G^y_{i,j} \leq +1$, which ensures a stable distillation in experiments.

### 4. Experiments

To simplify the story, we limit our attention to testing network transplanting operations. We do not discuss other related operations, e.g., the fine-tuning of category and task modules and the case in Fig. 1(a), which can be solved via traditional learning strategies.

We designed three experiments to evaluate the proposed method. In Experiment 1, we learned toy transplant nets by inserting adapters between middle layers of pre-trained CNNs. Then, Experiments 2 and 3 were designed considering the real application of learning a transplant net with two task modules (i.e., modules for object classification and segmentation) and multiple category modules. As shown in Fig. 4(b,d), we can divide the entire network-transplanting procedure into an operation sequence of transplanting category modules to the classification module and another operation sequence of transplanting category modules to the segmentation module. Therefore, we separately conducted the maps to the upper layers, and we consider this as **back-back-propagation**\(^1\). Tables 2 and 3 have exhibited the superior performance of the back-back-propagation.

![Figure 4](image-url)

**Figure 4.** (a) Teacher net; (b) transplant net; (c) two types of adapters; (d) two sequences of transplanting operations.
two sequences of transplanting operations in Experiments 2 and 3 for more convincing results.

Because our back-distillation strategy decreases the demand for training samples, we tested the learning of adapters with limited numbers of samples (i.e., 10, 20, 50, and 100 samples). We even tested network transplanting without any training samples in Experiment 1, i.e., optimizing the distillation loss without considering the task loss.

**Baselines:** We compared our back-distillation method (namely back-distill) with two baselines. All baselines exclusively learned the adapter without fine-tuning the task module for fair comparisons. The first baseline only optimized the task loss $\mathcal{L}(y_g, y'_g)$ without distillation, namely direct-learn. The second baseline is the traditional distillation [11], namely distill, where the distillation loss is CrossEntropy $(y_g, y'_g)$. The distillation was applied to outputs of task modules $g_s$ and $g_t$, because except for outputs, other layers in $g_s$ and $g_t$ did not produce features on similar manifolds. We tested the distill method in object segmentation, because unlike single-category classification, segmentation outputs had correlations between soft output labels.

**Network structures, datasets, & details:** We transplanted category modules to a classification module in the first two experiments, and transplanted category modules to a segmentation module in the third experiment. In recent studies, people usually extended the structure of widely-used VGG-16 net [21] to implement classification and segmentation in the third experiment. In our baseline experiments, we chose the VGG-16 and the FCN-8s as two task modules. Both branches are benchmark networks.

- The cross-entropy $\mathcal{L}(y_g, y'_g)$ measures the task loss.
- The task module $g_t$ is the teacher net for both classification and segmentation as a network.
- The first five layers of the FCN-8s and those of the VGG-16 share the same structure, we considered the first five layers (including two conv-layers, two ReLU layers, and one pooling layer) as the shared category module and regarded upper layers of the VGG-16 and the FCN-8s as two task modules. Both branches are benchmark networks.

We followed standard experimental settings in [15] to learn the FCN-8s for each category, which used the Pascal VOC 2011 dataset (with segmentation labels on 8498 PASCAL images collected by [9]). For object classification, we followed standard settings in [30] that used the PASCAL VOC images to learn CNNs for the binary classification of a single category from random images. Note that we only learned and merged teacher networks for five mammal categories, i.e., the cat, cow, dog, horse, and sheep categories. Mammal categories share similar object structures, which make features of a category transferable to other categories.

Now, we introduce adapter structures, as shown in Fig. 4(c). An adapter contained $n$ conv-layers, each followed by a ReLU layer ($n = 1$ or 3 in experiments). In addition, we inserted a “reorder” layer and a “re-scaling” layer in front of conv-layers in the adapter. The “reorder” layer randomly reordered channels of the features $x$ from the category module, which enlarged the dissimilarity between output features of different category modules.

We inserted the “reorder” layer to mimic feature states in real applications for fair evaluation. The “re-scaling” layer normalized the scale of features $x$ from the category module, i.e., $x^{ad} = \beta \cdot x$, for robust network transplanting. $\beta = \mathbb{E}_{I_t \in I_T}[[f_s(I)|f_I]/\mathbb{E}_{I_t \in I_T}[[f(I)|f_I]]$ is a fixed scalar. $I_T$ and $I_S$ denote the image set of the new category and the image set of categories that had been already modeled by the transplant net, respectively.

4.1. Exp. 1: Adding adapters to pre-trained CNNs

In this experiment, we conducted a toy test, i.e., inserting and learning an adapter between a category module and a task module to test network transplanting. Here, we only considered networks with VGG-16 structures [21] for single-category classification. These networks were strongly supervised using all training samples and achieved error rates of 1.6%, 0.6%, 4.1%, 1.6%, and 1.0% for the classification of the cat, cow, dog, horse, and sheep categories, respectively. Then, we learned two types of adapters (see Fig. 4(c)), which have been introduced before.

Because the classification output $y'$ is a scalar without neighboring outputs to provide correlations, we simply set $G'_s = 1$ without any gradient randomization. Instead, we used the revised dummy ReLU operations in Eqn. (4b) to ensure the value diversity of $D'_S, D'_T$ for learning. More specifically, we used $f^\text{dummy}_2 = f^\text{2nd}_\text{dummy}$ to compute expedient derivatives of ReLU operations in the task module, and used $f^\text{dummy}_1 = f^\text{1st}_\text{dummy}$ in the adapter$^4$. We set $\lambda = 10.0/\mathbb{E}_{I_t \in I_T}[[D_T]]$ for object classification in Experiments 1 and 2.

In Fig. 5, we compared the space of $fc8$ features, when we used our method and the direct-learn baseline, respectively, to learn the adapter with three conv-layers. The adapter learned by our method passed much stronger information to the final $fc8$ layer and yielded more diverse features. It demonstrates that our method better avoided problematic projections in Fig. 3 than the direct-learn baseline.

In Table 2, we compared our back-distill method with the direct-learn baseline when we inserted an adapter with $M$ filters. Each filter was a $3 \times 3 \times M$ tensor with a padding = 1 in Experiment 1 or a $1 \times 1 \times M$ tensor without padding in Experiments 2 and 3, to avoid changing the size of feature maps, where $M$ is the channel number of $f$.

2Each conv-layer in the adapter contained $M$ filters. Each filter was a $3 \times 3 \times M$ tensor with a padding = 1 in Experiment 1 or a $1 \times 1 \times M$ tensor without padding in Exps. 2 and 3, to avoid changing the size of feature maps, where $M$ is the channel number of $f$.

3Because we used the task module in the dog network as the generic task module $g_s$, we got $I'_S = I'_dog$.

4All derivative functions in Eqn. (4b) are only used for distillation, which will not affect gradient propagations from $\mathcal{L}(y, y')$.
Table 2. Error rates of classification when we insert \( n \) conv-layers with \( \text{ fc8 } \) features of different images, when our method learned the adapter. Red points correspond to \( \text{ fc8 } \) features of different images, when the adapter was learned by only using the task loss, i.e. the \text{ direct-learn } baseline. We visualize the first two principal components of \( \text{ fc8 } \) features. Because the \text{ direct-learn } baseline usually learned problematic many-to-one projections and projections to forgotten spaces (see Fig. 3), most information in \( h(x) \) could not pass through ReLU layers to the \( \text{ fc8 } \) layer. Therefore, given the adapter learned based on the \text{ direct-learn } baseline, units in the \( \text{ fc8 } \) layer were weakly triggered, and many-to-one projections decreased the diversity of \( \text{ fc8 } \) features.

![Figure 5. Comparison of the projected feature spaces. For each category, blue points indicate 4096-d \( \text{ fc8 } \) features of different images, when our method learned the adapter. Red points correspond to \( \text{ fc8 } \) features of different images, when the adapter was learned by only using the task loss, i.e. the \text{ direct-learn } baseline. We visualize the first two principal components of \( \text{ fc8 } \) features. Because the \text{ direct-learn } baseline usually learned problematic many-to-one projections and projections to forgotten spaces (see Fig. 3), most information in \( h(x) \) could not pass through ReLU layers to the \( \text{ fc8 } \) layer. Therefore, given the adapter learned based on the \text{ direct-learn } baseline, units in the \( \text{ fc8 } \) layer were weakly triggered, and many-to-one projections decreased the diversity of \( \text{ fc8 } \) features.](image)

Table 3 shows the performance when we transplanted the category module to a task module oriented to categories with similar structures. We tested our method with a few (10–100) training samples. When there were more than 50 training samples, our method yielded about a half classification error of the \text{ direct-learn } baseline.

![Table 3](image)
Experiment 2: transplanting to a classification module

| # of samples | cat    | cow    | horse   | sheep   | Avg.  |
|--------------|--------|--------|---------|---------|-------|
| 100          | direct-learn | 20.10  | 12.37   | 18.56   | 11.86  | 15.72 |
|              | back-distill | 9.79   | 5.67    | 8.25    | 4.64   | 7.09  |
| 50           | direct-learn | 22.68  | 19.39   | 19.07   | 14.95  | 19.07 |
|              | back-distill | 10.82  | 18.04   | 13.92   | 5.15   | 11.98 |

| # of samples | cat    | cow    | horse   | sheep   | Avg.  |
|--------------|--------|--------|---------|---------|-------|
| 20           | direct-learn | 31.96  | 37.11   | 39.69   | 35.57  | 36.08 |
|              | back-distill | 21.13  | 35.57   | 32.47   | 22.68  | 27.96 |

| # of samples | cat    | cow    | horse   | sheep   | Avg.  |
|--------------|--------|--------|---------|---------|-------|
| 10           | direct-learn | 41.75  | 37.63   | 44.33   | 33.51  | 39.31 |
|              | back-distill | 34.02  | 42.27   | 44.85   | 33.51  | 38.66 |

Table 3. (top) Error rate of single-category classification when we transplanted the classification module from a pre-trained dog network to the network of the target category in Experiment 2. The adapter contained three conv-layers. (bottom) Pixel accuracy of object segmentation when we transplanted the segmentation module from a dog network to the network of the target category in Experiment 3. The adapter contained a conv-layer and a ReLU layer.

Experiment 3: transplanting to a segmentation module

| # of samples | cat    | cow    | horse   | sheep   | Avg.  |
|--------------|--------|--------|---------|---------|-------|
| 100          | direct-learn | 76.54  | 74.60   | 81.00   | 78.37  | 77.63 |
|              | distill | 74.65  | 80.18   | 78.05   | 80.50  | 78.35 |
|              | back-distill | 85.17  | 90.04   | 90.13   | 86.53  | 87.97 |
| 50           | direct-learn | 71.30  | 74.76   | 76.83   | 78.47  | 75.34 |
|              | distill | 68.32  | 76.50   | 78.58   | 80.62  | 76.01 |
|              | back-distill | 83.14  | 90.02   | 90.46   | 85.58  | 87.30 |

| # of samples | cat    | cow    | horse   | sheep   | Avg.  |
|--------------|--------|--------|---------|---------|-------|
| 20           | direct-learn | 71.13  | 74.82   | 76.83   | 77.81  | 75.15 |
|              | distill | 71.17  | 74.82   | 76.05   | 78.10  | 75.04 |
|              | back-distill | 84.03  | 88.37   | 89.22   | 85.01  | 86.66 |
| 10           | distill | 70.46  | 74.74   | 76.49   | 78.25  | 74.99 |
|              | back-distill | 70.47  | 74.74   | 76.83   | 78.32  | 75.09 |
| 50           | direct-learn | 82.32  | 89.49   | 85.97   | 83.50  | 85.32 |

Table 4. (top) Error rate of single-category classification when we transplanted the classification module from a pre-trained dog network to the network of the target category in Experiment 2. The adapter contained three conv-layers. (bottom) Pixel accuracy of object segmentation when we transplanted the segmentation module from a dog network to the network of the target category in Experiment 3. The adapter contained a conv-layer and a ReLU layer.

4.4. Transplant to similar or dissimilar categories?

Theoretically, just like transfer learning, the task module should deal with a set of categories that have similar structures with the new category to ensure a high efficiency. In fact, identifying categories with similar structures is still an open problem. People manually define sets of similar categories, e.g. learning a task module for mammals and learning another task module for different vehicles.

In order to quantitatively evaluate the performance of transplanting to dissimilar categories, we designed new task modules for additional training. We considered the first four categories of the VOC dataset, i.e. aeroplane, bicycle, bird, and boat, to have dissimilar structures with mammals. In Experiment 2/Experiment 3, for the transplanting of a mammal category (let us take the cat for example) to a classification/segmentation module, we learned a “leave-one-out” classification/segmentation module to deal with all four dissimilar categories and all mammal categories except the cat.

Table 4 shows the performance of transplanting to a task module trained for both similar and dissimilar categories. Our method outperformed the baseline. Compared to the performance of transplanting to a task module oriented to similar categories in Table 3, transplanting to a more generic task modules for dissimilar categories hurt the segmentation performance but boosted the classification performance. It is because forcing a task module to handle dissimilar categories sometimes made the task module encode more generic and robust representations, while it may also let the task module ignore details of mammal categories.
5. Conclusions and discussion

In this paper, we focused on a new task, i.e., merging pre-trained teacher nets into a generic, modular transplant net with a few or even without training annotations. We discussed the importance and core challenges of this task. We developed the back-distillation algorithm as a theoretical solution to the challenging space-projection problem.

The back-distillation strategy significantly decreases the demand for training samples. Experimental results demonstrated the superior efficiency of our method. Our method without any training samples even outperformed the baseline with 100 training samples, as shown in Table 2(left).

The growth of a large transplant net for different categories and tasks can be divided into lots of elementary operations of network transplanting (see Fig. 1 and supplementary materials for more discussions). When the transplant net has been learned, we can optionally fine-tune task modules using training samples of multiple categories. Note that unlike the back distillation, the performance of fine-tuning depends on the number of training samples. Thus, given a few samples, whether an additional fine-tuning will increase or decrease the generality of the transplant net is a difficult question, and it requires sophisticated analysis in the future.

References

[1] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, 2016. 3

[2] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton. Large scale distributed neural network training through online distillation. In *ICLR*, 2018. 4

[3] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016. 3

[4] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas. Learning to learn without gradient descent by gradient descent. In *ICML*, 2017. 3

[5] Y.-M. Chou, Y.-M. Chan, J.-H. Lee, C.-Y. Chiu, and C.-S. Chen. Unifying and merging well-trained deep neural networks for inference stage. In *arXiv:1805.04980*, 2018. 2

[6] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Hay, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. In *arXiv:1701.08734*, 2017. 2, 3

[7] N. Frosst and G. Hinton. Distilling a neural network into a soft decision tree. In *arXiv:1711.09784*, 2017. 3, 4

[8] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation in backpropagation. In *ICML*, 2015. 3

[9] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 6

[10] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. β-vae: learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017. 3

[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Workshop*, 2014. 3, 6

[12] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulo. Deep neural decision forests. In *ICCV*, 2015. 3

[13] K. Li and J. Malik. Learning to optimize. In *arXiv:1606.01885*, 2016. 3

[14] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *arXiv:1712.00559*, 2017. 3

[15] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 6, 7

[16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. In *arXiv:1606.04671*, 2016. 2, 3

[17] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *NIPS*, 2017. 3

[18] R. Schwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. In *arXiv:1703.00810*, 2017. 4

[19] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. In *arXiv:1805.06370*, 2018. 2, 3

[20] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. outrageously large neural networks: the sparsely-gated mixture-of-experts layer. In *ICLR*, 2017. 3

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 6

[22] S. Srinivas and F. Fleuret. Knowledge transfer with jacobian matching. In *ICML*, 2018. 4, 13

[23] R. Vuroio, D.-Y. Cho, D. Kim, and J. Kim. Meta continual learning. In *arXiv:1806.06928*, 2018. 2, 3

[24] N. Wolchover. New theory cracks open the black box of deep learning. In *Quanta Magazine*, 2017. 4

[25] M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *NIPS TIML Workshop*, 2017. 3, 4

[26] T. Wu, X. Li, X. Song, W. Sun, L. Dong, and B. Li. Interpretable r-cnn. In *arXiv:1711.05226*, 2017. 3

[27] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning in a neural network. In *ICLR*, 2018. 2, 3

[28] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014. 3

[29] S. Zagoruyko and N. Komodakis. Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. In *arXiv:1612.03928*, 2017. 4
[30] Q. Zhang, Y. N. Wu, and S.-C. Zhu. Interpretable convolutional neural networks. In CVPR, 2018.

[31] Z. Zhong, J. Yan, and C.-L. Liu. Practical network blocks design with q-learning. In AAAI, 2018.

[32] Z.-H. Zhou and J. Feng. Deep forest: Towards an alternative to deep neural networks. In IJCAI, 2017.

[33] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In ICLR, 2017.

[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In arXiv:1707.07012, 2017.
Computation of gradients w.r.t. the distillation loss

In order to optimize the distillation loss, we need to compute $\frac{\partial D'(\theta_h)}{\partial \theta_h}$, i.e. $\frac{\partial D'(\theta)}{\partial \theta}$ with respect to the following $D'(\theta)$. 

$$D'(\theta) \overset{\text{def}}{=} G' \frac{\partial y'}{\partial x(n)} \cdots \frac{\partial x(m+1)}{\partial x(m)} \bigg|_{x=x(m)} = f'_\text{conv} \circ f'_{\text{dummy}} \circ f'_{\text{avg}} \circ \cdots \circ f'_{\text{conv}}(G'y')$$

Thus, we first explore the close-form formulation of the function $D'(\theta)$. As shown in the above equation, we can transform the back-propagation process for computing $D'(\theta)$ as a number of cascaded functions $f'_\text{conv}, \ldots, f'_{\text{avg}}, f'_{\text{dummy}}, f'_{\text{conv}},$ which are derivatives of $f_{\text{conv}}, \ldots, f_{\text{avg}}, f_{\text{dummy}}, f_{\text{conv}}$. Since we have formulated $f'_{\text{dummy}}$ in the manuscript and it is easy to obtain $f'_{\text{avg}}$, we mainly focus on the formulation of $f'_{\text{conv}}$. 

In general, it is not difficult to derive the derivative of any convolution operation. Here, we focus on the most common case, i.e. the convolution operation with a padding $p$ and a stride of 1. Given a tensor $x \in \mathbb{R}^{M \times M \times D}$ and $C$ convolutional filter with weights $w \in \mathbb{R}^{m \times m \times D \times C}$ and a bias term $b \in \mathbb{R}^C$, the convolution can be written as $y = x \otimes w + b$. For VGG networks, people usually set $m = 2p + 1$. $w$ can further absorb $b$ by adding the $(D+1)-th$ channel to $w$ and adding the $(D+1)-th$ channel to $x$ with $x_{:, :, D+1} = 1$. Thus, we can obtain

$$y = x \otimes w$$

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y} \otimes W$$

where $W \in \mathbb{R}^{m \times m \times C \times D}$ and $W_{i,j,k,l} = w_{m+1-i,m+1-j,l,k}$. Thus, we can write the derivative of $f_{\text{conv}}$ as

$$f'_{\text{conv}}(G') = G' \otimes W$$

In this way, we obtain the close-form formulation of the function $D'(\theta)$. We can easily compute $\frac{\partial D'(\theta)}{\partial \theta}$ using the chain rule of back propagation. We can consider this process as a back-back-propagation.

About the case of using an enlarged $G'y'$

When we use an enlarged pseudo-gradient $G'y' \in [-1, +1]^{S \times S \times 1}$, we can obtain an enlarged gradient map $D'_{\Sigma}$. As discussed above, the computation of $D'_{\Sigma}$ can be considered as a number of cascaded functions $f'_{\text{conv}}, \ldots, f'_{\text{avg}}, f'_{\text{dummy}}, f'_{\text{conv}}$ with the input $G'y'$, which are quite similar to the forward propagation in the neural network.
Visualization of network structures used in three experiments

Adapter with three conv-layers and three ReLU layers

This structure is used in Experiments 1 and 2

This structure is used in Experiments 1 and 3

a classification module in Exp. 1 or a segmentation module in Exp. 3

Category module

Task module
How to learn a large transplant network

In the paper, we limit our attention to the core technique of back distillation for network transplanting and do preliminary experiments to demonstrate the effectiveness of the proposed method. Here, we would like to explain how to use the proposed back-distillation algorithm to gradually grow a large transplant network. The basic idea of growing a large transplant network has been shown in Figure 1 of the paper. We can divide the complex procedure of building a large transplant network into elementary operations of network transplanting.

Generally speaking, there are two types of operations during the learning of the transplant network, i.e. adding a new task module and adding a new category module.

**Adding a new task module:** Compared to adding category modules, adding task modules is relatively easier. There are two ways to add a task module.

Firstly, given a single or multiple pre-trained category modules and training samples of the category/categories, we can directly learn a new task module upon features of the category module(s). This learning process has been shown as “Step 1” in Figure 1, which follows traditional learning strategy without network transplanting.

Secondly, when the specific network is pre-trained with a single category module and multiple task modules, we can transplant the category module to a generic task module in the target transplant network. In this case, task modules (those do not correspond to ) will automatically be added to the transplant network, although these task modules only connect with a single category module. This case has been shown as “Step 2” in Figure 1, in which the Task-3 module is added to the transplant network when we connect the category-D module to the Task-2 module of the transplant network.

**Adding a new category module:** We can also divide the insertion of a category modules into the following two cases. The first case is the traditional network transplanting that is introduced in this paper. The second case is to build more connections between category modules and task modules that have already been contained by the transplant network. This case is shown as “Step 3” in Figure 1. We learn new adapters to connect existing category modules and task modules via network transplanting.

Comparing our back-distillation with the Jacobian distillation

The loss for the Jacobian distillation is similar to Equation (3). There are two essential differences between them, which make the Jacobian distillation not applicable to network transplanting. (i) The Jacobian distillation does not balance magnitudes of neural activations between the category module and the task module, which significantly increases the difficulties of network transplanting. (ii) More crucially, the Jacobian distillation uses real gradients of the network instead of using pseudo-gradients. However, when we fixed the upper task module during network transplanting, the task module usually blocks most signals during the forward propagation. Thus, during the back propagation, real gradients usually cannot pass ReLU layers in the task module to produce informative Jacobians for distillation.

The core challenge

To clarify the challenge of learning the adapter, we will compare the following two cases, i.e. 1) learning both the adapter and the task module and 2) learning the adapter but fixing the task module . The traditional method of directly optimizing the task loss can easily solve the first case but will be hampered in the second case.

**Case 1, Learning both the adapter and the task module:** This case corresponds to the traditional problem of deep learning. We initialize the task module with random parameters, so in the first epoch of learning, input features will produce random activations in layers of both and , and positive neural activations will pass through ReLU layers to make random predictions . Successfully passing information to the final output is quite important, because we can obtain gradients of the task loss to optimize and . In this way, both and can be well learned.

**Case 2, learning the adapter but fixing the task module:** Unlike Case 1, we use a pre-trained task module , and we fix its parameters during the learning process. We only initialize the adapter with random parameters. However, as shown in Fig. 3(left) in the paper, the task module has vast forgotten space of its input feature, and can only pass very specific features to the final output. In other words, cannot pass most information of ’s features to the final output in early epochs. Thus, we will not obtain informative gradients to optimize parameters in .