Partial Network Cloning
—*Supplemental Material*—

Jingwen Ye Songhua Liu Xinchao Wang [†] [*]
National University of Singapore
{jingweny,xinchao}@nus.edu.sg, songhua.liu@u.nus.edu

In this document, we provide the supplementary materials that cannot fit into the main manuscript due to the page limit, including more details and analysis on the proposed PNC model, as well as more experimental results.

1. More Details of PNC

Here we give more details on the proposed partial network cloning framework, including the estimation of the local model set, the extended PNC to the multiple sources case, as well as the overall algorithm.

1.1. Computing the Local Model Set

In order to locate the transferable module $\mathcal{M}_f$ in the source network, we train a model set containing N small local models $\mathcal{G} = \{g_i\}^{(N)}$ to model the source $\mathcal{M}_s$ in the D_t neighborhood, and then use the local model set as the surrogate as $\mathcal{G} \approx \mathcal{M}_s|_{D_t}$. As mentioned in the manuscript, $\mathcal{G}$ is computed as:

$$\mathcal{G} \leftarrow \Big\{ g_i \Big| g_i = \arg\min_{g_i} \frac{1}{|B|} \sum_{b \in B} \Pi_b \cdot \big\| \mathcal{M}_s(b \cdot x_i) - g_i(b) \big\|^2 + \Omega(g_i), i \in \{1, 2, ..., N\} \Big\}. \tag{1}$$

For each x_i, we separate the images into $H \times H$ patches as $x_i = x_i^1 \circ x_i^2 \circ \cdots \circ x_i^{H^2}$, and then conduct the mask-based perturbations on x_i with a binary mask b, where $b = \{0, 1\}^{H^2}$. Specifically, the localized x_i is in the form of:

$$(b \cdot x_i)^h = \begin{cases} x_i^h & b^h = 1 \\ Mean(x_i^h) & b^h = 0 \end{cases}, \tag{2}$$

where $Mean(\cdot)$ is for computing the mean RGB values of a given region and then applying these fixed values to all the pixels in that region. Π_b is for measuring the perturbed similarity with the original x_i, which is:

$$\Pi_b = e^{-\sqrt{\frac{\cos(b, \mathbf{1})}{\sigma^2}}}, \tag{3}$$

where we set $\sigma = 0.25$. In this way, $\mathcal{G}$ is computed by the least square method as:

$$\begin{aligned} &\theta_i = (X_i^T X_i)^{-1} X_i Y_i, \\ &s.t. \quad X_i = \big\{ \hat{x_i} \big| \hat{x_i} = b \cdot x_i, b \in B \big\} \\ &\qquad Y_i = \big\{ \hat{y_i} \big| y_i = \Pi_b \cdot \mathcal{M}_s\big(\hat{x_i}\big), \hat{x_i} \in X_i \big\} \end{aligned} \tag{4}$$

where θ_i is an $H \times H \times C$ weight of g_i, and C is the dimension of the final FC layer of $\mathcal{M}_s$. Since g_i is computed based on the augmented neighbor of x_i, we do not need to compute g_i for all the samples of D_t ($N < |D_t|$). Then for the local model set $\mathcal{G}$, the weights are $\Theta = \theta_1 \circ \theta_2 \circ ... \circ \theta_N$.

1.2. Multi-Network Cloning

The proposed PNC framework is also capable of dealing with the task of cloning from *multiple sources*. Here we discuss two scenarios on multi-network cloning: cloning *different* functionalities from different source models, and cloning the *same* functionalities from different source models.

1.2.1 Cloning for Different Functionalities

The source model set is given as $\mathcal{M}_s = \{\mathcal{M}_s^0, \mathcal{M}_s^1, ..., \mathcal{M}_s^{P-1}\}$, where each $\mathcal{M}_s^\rho$ ($0 \le \rho < P$) serves for cloning the functionality t_s^ρ. We partially clone P transferable modules $\mathcal{M}_f = \{\mathcal{M}_f^0, \mathcal{M}_f^1, ..., \mathcal{M}_f^{P-1}\}$ with the P selection functions $M = \{M^0, M^1, ..., M^{P-1}\}$ and Insertion positions $R = \{R^0, R^1, ..., R^{P-1}\}$. The cloned network $\mathcal{M}_c$ can then be denoted as:

$$\begin{aligned} \mathcal{M}_c &\leftarrow Clone(\mathcal{M}_t, M, \mathcal{M}_s, R) \\ &\leftarrow Insert_{\rho=0}^{P}(\mathcal{M}_t, \mathcal{M}_f^\rho, R^\rho). \end{aligned} \tag{5}$$

In this way, a total of P functionalities are added to $\mathcal{M}_c$ by partially cloning from P source networks.

1.2.2 Cloning for the Same Functionality

As for a certain functionality T_s, there may exist multiple pre-trained models available for the partial network cloning.

[*†] Corresponding author.

Our goal is thus to select one source network with the best cloning performance to carry out the partial network cloning:

$$\begin{aligned} \mathcal{M}_c^* &= \arg\max_{\mathcal{M}_c^\rho} Acc(\mathcal{M}_c^\rho) \\ where \quad & \mathcal{M}_c^\rho \leftarrow Clone(\mathcal{M}_t, M, \mathcal{M}_s^\rho, R) \\ and \quad & \rho \in \{0, 1, ..., P-1\}. \end{aligned} \tag{6}$$

The key of cloning the same functionality from multiple source networks is, therefore, to determine which source may contribute to the best cloning performance on the target network. A straightforward solution is to sequentially apply PNC from each source $\mathcal{M}_s^\rho$ to the target network, so as to obtain each $\mathcal{M}_c^i$ for the accuracy evaluation. However, this straightforward solution scales up as the number of the sources increases.

As discussed in the manuscript, $\mathcal{G} \leftarrow \mathcal{M}|_D$ can be computed in advance for each pre-trained model, which also potentially serves as a tool for measuring model distances. Here, we use the similarity metrics $Sim(\cdot|\cdot)$ as an approximate indicator to select the optimal source $\mathcal{M}_s^*$ firstly, and then apply PNC on the selected $\mathcal{M}_s^*$, which directly reduces the learning process in PNC to $1/P$. To be specific, the optimal $\mathcal{M}_s^*$ is:

$$\begin{aligned} \rho^* &= \arg\max_\rho Sim\big(\mathcal{M}_c^\rho|D_t, \mathcal{M}_t|D_t\big), \\ &= \arg\max_\rho \frac{\Theta_s^\rho \cdot \Theta_t}{\|\Theta_s^\rho\| \cdot \|\Theta_t\|} \\ s.t. \quad & \rho \in \{0, 1, ..., P-1\} \end{aligned} \tag{7}$$

where Θ^ρ is the weight of the local model set computed on $\mathcal{M}_s^\rho$ in the D_t neighborhood, and Θ^ρ is the one computed on the target network. Due to different FC layers of the target and the source models, alignment needs to be done for these two weights. We follow the work of ZEST [2] to carry out the alignment.

In this way, the optimal $\mathcal{M}_c$ takes the form of

$$\mathcal{M}_c^\rho \leftarrow Clone(\mathcal{M}_t, M, \mathcal{M}_s^{\rho^*}, R), \tag{8}$$

where we only partially clone one transferable module from one source network, enabling a more efficient PNC.

1.3. Algorithm

The complete algorithm of PNC is given in Alg. 1, where the 'Pre-process' is to get the local model set $\mathcal{G}$ and 'New Functionality Addition' is to clone part of the source to the target.

Algorithm 1 Partial Network Cloning

Require: $\mathcal{M}_s$: source model; $\mathcal{M}_t$: target model; D_t: samples related to the to-be-clone functionality;
Ensure: $\mathcal{M}_c$: target network added with the new functionality.

———————***Pre-process***———————
For each $x_i \in D_t$, segment it into $H \times H$ patches as $x_i = x_i^1 \circ x_i^2 \circ \cdots \circ x_i^{H^2}$;
Generate the binary mask set B with random 0 and 1;
Random sample the regions several times as B;
Compute the local model set $\mathcal{G}$ by Eq. 1;
———————***New Functionality Addition***———————
Divide $\mathcal{M}_s$ and $\mathcal{M}_t$ into L blocks.
Set loss convergence value set: $loss \leftarrow \{\}$
for $R : L-1 \rightarrow 0$ **do**
    Initialize adapter $\mathcal{A}^R$, classifier $\mathcal{F}_c^R$ and selector M;
    while not convergence **do**
        Build $\mathcal{M}_f \leftarrow M \cdot \mathcal{M}_s^{[R:L]}$
        Build $\mathcal{M}_c \leftarrow \mathcal{M}_t^{[0:R]} \circ \{\mathcal{M}_t^{[R:L]}, \mathcal{A}^R \circ \mathcal{M}_f\} \circ \mathcal{F}_c^R$
        Input $B \cdot x$ to $\mathcal{M}_s^{[0:R]} \circ \mathcal{M}_f$, B to $\mathcal{G}$ to calculate $\mathcal{L}_{loc}$;
        Input $B \cdot x$ to $\mathcal{M}_c$, B to $\mathcal{G}$ to calculate $\mathcal{L}_{ins}$;
        Minimize $\mathcal{L}_{loc} + \mathcal{L}_{ins}$ to update $\mathcal{A}^R$ $\mathcal{F}_c^R$ and M;
        Collect the loss convergence value $loss^R$;
    end while
    Update loss as $loss$.append($loss^R$);
end for
Binarize M as '0/1' mask;
Determine insertion position $R^* = \arg\min loss^R$;
Build transferable module $\mathcal{M}_f \leftarrow M \cdot \mathcal{M}_s^{[R^*:L]}$;
Return $\mathcal{M}_c \leftarrow \mathcal{M}_t^{[0:R^*]} \circ \{\mathcal{M}_t^{[R^*:L]}, \mathcal{A}^{R^*} \circ \mathcal{M}_f\} \circ \mathcal{F}_c^{R^*}$.

2. Efficiency and Sustainability Analysis

The proposed partial network cloning is indeed efficient and sustainable in the following senses:

- **Data dependency.** PNC has decreased the data dependency to about 30%, compared with the normal continual learning setting; when compared with the network with scratch training, it reduces to $30\%/C$, where C is the total number of classes.
- **Training complexity.** PNC proposes an efficient training strategy. Firstly, only the parameters of $\mathcal{A}$, $\mathcal{F}_c$ and M are updated with only a few (about 10) epochs; secondly, we design a simple but effect searching strategy for determining the best insertion position.
- **Storage.** Here we discuss the storage from two aspects, offline and online. Offline storage is the exact storage of $\mathcal{M}_c$, which could enable offline usage after downloading. And offline storage is the storage for the part of parameters other than the part that could be used from model zoo (pre-trained source and target), which enables online usage together with model zoo.
- **Recoverable.** PNC proposes to add new functionality to the target network in a recoverable way. In other words, we can remove the added functionality once we no longer need it and recover the original target network.

Table 1. Experimental settings on different source target pairs and on different datasets, including the searching steps ('S Step'), fine-tune epochs per search ('Epoch'), percentage of the to-be-cloned data set ('Data Size'), the number of mask B ('Aug Num') and masking rate.

Dataset	Source	Target	Settings				
			S Step	Epoch	Data Size	Aug Num	Masking Rate
MNIST	LeNet5	LeNet5	3	10	0.1	100	0.3
CIFAR10	ResNet-18	Plain CNN	3	10	0.3	100	0.3
CIFAR10	ResNet-18	ResNet-18	5	10	0.3	100	0.3
CIFAR10	ResNet-50	ResNet-18	5	10	0.3	100	0.3
CIFAR100	ResNet-18	ResNet-18	5	20	0.3	100	0.3
CIFAR100	ResNet-18	MobileNetV2	5	20	0.3	100	0.3
CIFAR100	ResNet-18	ShuffleNetV2	5	20	0.3	100	0.3
TinyImageNet	ResNet-18	ResNet-18	5	20	0.3	1000	0.3

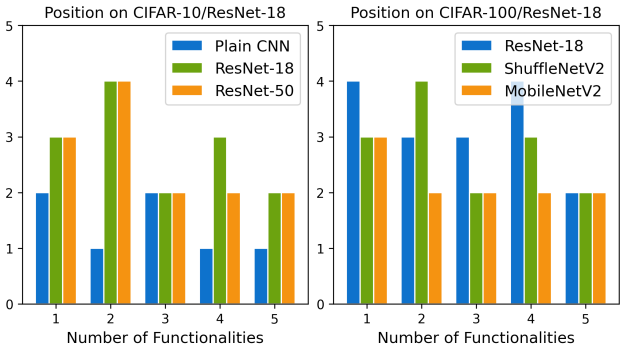

Figure 1. The final searching results on CIFAR-10 and CIFAR-100 datasets. ResNet-18 is set as the base target network.

3. More Experiments

3.1. Experimental Setting

The parameter settings are listed in Table 1, where the experiments are done on various datasets (MNIST, CIFAR-10, CIFAR-100 and TinyImageNet datasets) and architectures (LeNet, plain CNN, ResNet-18, ResNet-50, MobileNetV2 and ShuffleNetV2.).

3.2. Ablation Study

Searching Result of Position R For the source and the target networks in the same architecture, the searching step is equal to the pre-defined number of blocks L (we use the most commonly used split). As for the heterogeneous network pairs, where the source network is divided into L_s blocks and the target network is divided into L_t blocks, the searching steps are set to be L_t, and for each searching step $R_t \in \{0, 1, ..., L_t - 1\}$, the corresponding R_s is set as $R_s = [R_t L_s / L_t]$. Fig. 1 shows the searching result of each R for each settings, where we show the results on CIFAR-10 and CIFAR-100 datasets, and both the source and the target are in the same network architecture–ResNet-18. The searching results with larger R mean the target network shares more of its feature extractor layers with the transferable module. Thus, as can be observed form Fig. 1, when the number of to-be-cloned functionalities increase, the searching results with R becomes smaller, which is mainly because that the transferable module needs to transfer more knowledge from the source (smaller R also means larger transferable module).

Table 2. Experiments on different data size for training on CIFAR10 dataset with ResNet-18 as the base network.

Data Size	Ori. Acc	Tar. Acc	Avg. Acc
1% D_t	60.0	69.5	61.6
10% D_t	85.9	85.8	85.9
30% D_t	94.4	95.8	94.6
60% D_t	94.5	96.2	94.8
100% D_t	94.3	94.9	94.4

Size of data for training. As we have stated in the experimental setting, we only use 30% of the target data as the training data, which shows the data efficiency of the proposed PNC. Here, we conduct the experiments on how the size of the data influences the final cloning performance. As is shown in Table 2, more data included for training would contribute to better cloning performance, but this promotion becomes smaller when 30% data is used for training. The main reason is that we augment the training samples in its neighbor, which has better representative ability for the source network. As larger data would definitely increase the training cost, we set the ratio for training to be 30% in the rest of the experiments.

Number of to-be-cloned functionalities. The position searching results on different numbers of functionalities are depicted in Fig. 3. Here we display the results on Table 4. We partially clone 1,2,3,4 and 5 functionalities from the source, where cloning 5-label functionality is to transfer all the functionalities from the source. PNC does good when

Table 3. Experimental results of cloning with different functionalities from different source models. The new functionalities are added sequentially numbered as 1-9 with 1-label functionality addition ('s') and 5-label functionality addition ('m').

Methods	Params	Acc-1	Acc-2	Acc-3	Acc-4	Acc-5	Acc-6	Acc-7	Acc-8	Acc-9	Acc-10	Avg.
Pre-trained	$\sim 113.3M$	84.8	80.5	80.7	75.2	80.6	83.0	82.8	77.9	85.0	85.5	81.6
PNC-s1	$\sim 13.8M$	84.5	81.9	-	-	-	-	-	-	-	-	84.3
PNC-s2	$\sim 16.3M$	83.1	79.5	83.8	-	-	-	-	-	-	-	82.9
PNC-s3	$\sim 18.8M$	83.4	75.6	80.5	80.2	-	-	-	-	-	-	82.3
PNC-s4	$\sim 20.3M$	81.9	74.9	79.3	78.5	80.2	-	-	-	-	-	80.9
PNC-s5	$\sim 21.8M$	80.6	75.8	76.1	77.6	79.5	84.4	-	-	-	-	80.0
PNC-s6	$\sim 23.3M$	79.9	74.2	77.4	77.6	78.8	82.5	85.2	-	-	-	79.7
PNC-s7	$\sim 24.8M$	80.0	75.2	77.9	75.8	77.9	83.7	83.5	80.9	-	-	79.7
PNC-s8	$\sim 26.3M$	79.2	73.2	76.2	75.2	75.1	82.0	83.4	79.2	86.0	-	79.0
PNC-s9	$\sim 27.3M$	73.8	69.7	70.9	69.3	74.3	80.8	76.3	76.4	85.3	87.0	75.2
PNC-m1	$\sim 13.9M$	84.6	80.8	-	-	-	-	-	-	-	-	83.3
PNC-m2	$\sim 17.1M$	83.4	79.7	82.7	-	-	-	-	-	-	-	82.3
PNC-m3	$\sim 19.8M$	80.6	77.1	81.4	77.0	-	-	-	-	-	-	79.3
PNC-m4	$\sim 22.4M$	78.4	77.3	80.5	77.4	80.6	-	-	-	-	-	78.8
PNC-m5	$\sim 25.0M$	78.0	76.8	79.7	75.2	78.9	83.1	-	-	-	-	78.5
PNC-m6	$\sim 28.2M$	77.4	77.2	77.7	74.6	79.2	82.7	83.0	-	-	-	78.7
PNC-m7	$\sim 30.7M$	75.5	73.4	76.1	73.8	73.3	82.8	82.9	80.6	-	-	77.1
PNC-m8	$\sim 33.9M$	75.1	73.1	72.1	72.5	74.4	81.0	81.4	75.3	84.4	-	76.4
PNC-m9	$\sim 37.1M$	70.9	66.5	68.9	70.1	66.8	80.5	80.2	75.5	83.8	86.6	74.6

Table 4. Experiments on different numbers of to-be-cloned functionalities on CIFAR10 dataset with ResNet-18 as the base.

Functions	Ori. Acc	Tar. Acc	Avg. Acc
1	94.4	95.8	94.6
2	94.2	95.3	94.5
3	93.7	94.5	94.0
4	93.1	94.6	93.8
5 (Full)	93.3	94.2	94.1

Table 5. Experiments on cloning with the same functionality from the source networks with different architectures.

	Source		
Metrics	CNN	ResNet-18	ResNet-50
Source Acc	88.1	95.9	96.2
Ori. Acc	91.7	94.4	94.0
Tar. Acc	93.4	95.8	93.3
Avg. Acc	92.0	94.6	93.9

partially transferring a small number of functionalities from the source ('Function 1'), while showing its adorable performance even transferring the full set of the source ('Function 5').

3.3. Experiments on Multi-source Cloning

Here we provide the experimental results on cloning from multiple source models. This part of experiment is conducted on CIFAR-100 datasets with ResNet-18 as the base architecture for the target network $\mathcal{M}_t$.

Cloning with different functionalities. We separate the classification task of CIFAR-100 dataset uniformly into 10 sub-tasks, each trained on ResNet-18 as $\mathcal{M}_1 \sim \mathcal{M}_{10}$. We choose to take $\mathcal{M}_1$ as the target network, and the rest as the source networks. The experimental results are depicted in Table 3. As is shown in the table, the new network obtained by PNC is much smaller than the original pre-trained networks, thus providing a resource friendly functional addition method. And it could also be observed that when the number of source models increases, the average accuracies drop slightly. As such, cloning from too many source models is, in reality, not a good way for adding new functionalities to the target model. An efficient way to apply PNC is to choose the pre-trained target network that has the most similar functionalities as required and then choose the least number of source models to transfer the rest functionalities.

Cloning with the same functionality. Here we separate the CIFAR-10 dataset uniformly into 2 sub-tasks, one is trained on ResNet-18 as the target, the other subset is trained on plain CNN, ResNet-18 and ResNet-50 separately as three different source networks. We choose to clone one-label task from the source networks. The corresponding results are depicted on Table 5. As can be observed from the table that, when the similarity score increases, the PNC performance has been promoted, which has proved the ef-

Table 6. Experiments on incremental learning on split CIFAR-100 dataset, where the dataset is split into 10 sub-tasks.

Method	Recov. ?	Extra Params	Avg. Acc
Grow [7]	yes	1.3×	54.7
APD [6]	no	1.3×	57.5
CPG [1]	no	1.3×	60.1
GROWN [5]	yes	1.2×	60.8
FFNB [3]	yes	0.5×	68.1
DER [4]	yes	0.8×	72.5
PNC (Ours)	yes	0.1×	73.6

fectiveness of the source model selection strategy.

3.4. More Comparisons with Continual Learning

We note that the most related work to the proposed PNC is the architecture-based continual learning, which aims at minimizing the inter-task interference via newly designed architectural components. Following the previous work [5], the methods listed for comparison can be further divided into two steams:

- **Grow only:** This stream of methods only grow the model for each task without effect on the pre-trained backbone part during training, which also means recoverable when simply abandoning the new-grow architectures.
- **Grow-and-prune:** This stream of methods gradually grow and prunes the backbone model for each new task, which also mean unrecoverable.

We compare the proposed method in two ways, one is to sequentially add new functionalities and the other is to sequentially remove these functionalities again. Thus, the performance is evaluated in both the class-*incremental* and the class-*decremental* manners (whether it is recoverable? 'Recov.?'). The comparative results are depicted in Table 6. It could be observed that, the proposed PNC obtain the best performance on incremental learning, with its operation-recoverable property and the least extra parameters (only need new parameters for the light-weight adapt modules).

References

[1] Steven C. Y. Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, picking and growing for unforgetting continual learning. In *Neural Information Processing Systems*, 2019. 5

[2] Hengrui Jia, Hongyu Chen, Jonas Guan, Ali Shahin Shamsabadi, and Nicolas Papernot. A zest of lime: Towards architecture-independent model distances. In *International Conference on Learning Representations*, 2022. 2

[3] Hichem Sahbi and Haoming Zhan. Ffnb: Forgetting-free neural blocks for deep continual learning. In *BMVC*, 2021. 5

[4] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. 2021. 5

[5] Li Yang, Sen Lin, Junshan Zhang, and Deliang Fan. Grown: Grow only when necessary for continual learning. *ArXiv*, abs/2110.00908, 2021. 5

[6] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. *arXiv: Learning*, 2020. 5

[7] Xin Yuan, Pedro H. P. Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. *ArXiv*, abs/2007.15353, 2021. 5