Supplementary Material: Neural Substitution for Branch-level Network Re-parameterization

In supplementary material, our PyTorch implementation is provided to enhance understanding of our method. We also offer details of our hyper-parameter setting in the experiments.

A PyTorch Implementation

A.1 Guided Activation

```python
import torch
from torch.nn.functional import relu


def guided_activation(feature: torch.Tensor) -> torch.Tensor:
    """
    Guided activation function for branch-level neural
        substitution.

    Args:
        feature (‘torch.Tensor’): Output features of multiple
             convolutions.
        The shape is (N,C,H,W), representing number of
            features(not batch size), channels, height, and
            width, respectively.
    Returns:
        ‘torch.Tensor’: Guided output features.
    """

    gathered_feature = feature.mean(dim=0)
    gathered_feature = relu(gathered_feature)

    guided_map = (gathered_feature != 0).float()
    feature = torch.mul(feature, guided_map.unsqueeze(-1))

    return feature
```

A.2 Stochastic Neural Substitution

```python
import torch
from torch import nn


def neural_substitution(x: torch.Tensor, convolutions: nn.
    ModuleList) -> torch.Tensor:
    """
    Stochastic neural substitution for branch-level
        connectivity

    Args:
        x (torch.Tensor): Input features of multiple
            convolutions.
        The shape is (N,C,H,W), representing number of
            features(not batch size), channels, height, and
            width, respectively.
        convolutions (nn.ModuleList): The list of multiple
            convolutions.
    Returns:
        `torch.Tensor': The substituted features that have
            passed multiple convolutions.
    """

    N, C, H, W = x.size()
    n_conv = len(convolutions)
    out_features = list()

    for conv_module in convolutions:
        out_features.append(conv_module(x))

    out_features = torch.cat(out_features, dim=0)
    out_features = out_features[torch.randperm(n_conv * N)]

    out_features = out_features.reshape(n_conv, N, C, H, W).
        sum(0)
    return out_features
```

B Details of Hyper-parameter Setting

We used four NVIDIA A5000 GPUs for training the ImageNet dataset and a single GPU for the other datasets. The versions of PyTorch and Python are 2.2.1+cu121 and 3.10.12, respectively.

Table 1. Details of the hyperparameter settings for the ImageNet and CIFAR100 datasets. Note that in CIFAR100 and imageNet, MobileNetV1 and MobileOne utilize 128 and 512 batch sizes. The 9 datasets mean that the experiment setting of Table 4.

Dataset	CIFAR100	ImageNet	9 datasets
Epochs	100	100	30
Batch size	2048	1024	256
Optimizer	LAMB	LAMB	AdamW
Weight decay	$1.0e^{-2}$	$1.0e^{-2}$	$1.0e^{-2}$
LR(Learning rate)	$3.5e^{-3}$	$3.5e^{-3}$	$3.5e^{-3}$
Warmup epoch	5	3	5
Warmup LR	$1.0e^{-5}$	$1.0e^{-4}$	$1.0e^{-5}$
Min LR	$1.0e^{-6}$	$1.0e^{-6}$	$1.0e^{-5}$
Image size	$3 \times 32 \times 32$	$3 \times 224 \times 224$	$3 \times 224 \times 224$
Label smoothing	0.1	0.0	0.0
Rand Augment	X	7 / 0.5	7 / 0.5
Auto Augment	CIFAR10 policy	X	X
Cutmix	0.0	1	0.0
Mixup	0.0	0.1	0.0
Loss	Cross Entropy	Binary Cross Entropy (0.2)	Cross Entropy
Color Jitter	0.0	0.4	0.0
Train interpolation	bicubic	random	bicubic
Test interpolation	bicubic	bicubic	bicubic
Test crop ratio	1.0	0.95	1.0
Stoch. Depth	0.15	0.05	0.0