# Convolutional Network with Sub-Networks

**Ninnart Fuengfusin**
*Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology,*
*2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan*
**Hakaru Tamukoh**
*Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology,*
*2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan*
*E-mail: fuengfusin.ninnart553@mail.kyutech.jp, tamukoh@brain.kyutech.jp*
*https://www.brain.kyutech.ac.jp/~tamukoh/*

**Abstract**

We propose a convolutional network with sub-networks (CNSN), a convolutional neural network (CNN) that can be detached into sub-models on fly. Due to a conventional design of CNN, shapes of feature map are varied throughout the model. Therefore, the hidden layer within CNN may not directly process the input image without any modifications. To address this problem, we propose a *step-down convolutional layer,* a convolutional layer which acts as an input layer for the sub-model. The *step-down convolutional layer* reshapes the input image to a preferred representation to the sub-model. To train CNSN, we treat the base-model and sub-models as different models. We separately forward- and back-propagate each model. By using *multi-model loss*, a linear combination of losses from base- and sub-models, we can update weights that can be utilized in both base- and sub-models.

*Keywords*: Convolutional Neural Networks, Supervised Learning, Model Compression.

## 1. Introduction

In recent years, a convolutional neural network (CNN) has been achieved state-of-art results in varied tasks from an image recognition[1], sematic segmentation[2], and object detection[3] tasks. However, when comparing the CNN with a multilayer perceptron (MLP) with same amount of weight parameters, the down-side of CNN is it significantly utilizes more multiply–accumulate operations (MAC) than MLP. This may lead into a bottleneck when deploying a CNN with a mobile device that has lower computational capacity. To address this problem, there are a several research directions attempt to address this problem from a structure pruning[4], network sliming[5], and early-exit network[6].

In this research, we focus on the network sliming direction. The network sliming focuses on create a neural network model that can be detached into smaller models during inference. In our previous work in this direction, we purposed Network with Sub-Networks[7] (NSN) that introduces a layer-wise detach ability to MLP. By detaching weight layers, this allows NSN to operate with less amount of MAC, therefore it decreases an overall latency of model with the trade-off in the model performance. However, to apply the methods in NSN to CNN, there are three compatibility issues. The first issue is NSN requires a targeted model to contain a same shape of feature map throughout the model. This creates a major constraint to the CNN model. The CNN cannot contain any spatial reduction layers, for example pooling layers or convolutional layers without padding. Without any spatial reduction layers, this may lead the CNN to operate with higher MAC than the layer detaching can reduce. The second issue is NSN requires to train $N+1$ models in parallel, where $N$ is a number of hidden layers. This indicates that NSN method is not feasible with the recent CNN models that contains more than hundred hidden layers. The last issue is NSN require to manually control the gradients across base- and sub-models. This causes the high complexity in programming.

In this research, we propose CNSN to address all mentioned issues from our previous work. CNSN allows the usage of different shapes of feature map by attaching a *step-down convolutional layer* as the input layer of sub-models. This layer pre-processes the input image to a preferred representation to the sub-model. To allow CNSN to operate more depth CNN models, CNSN reduces a number of sub-models by allowing only few selected sub-models to be detached with. Instead of manually control the gradients, we introduce a *multi-model loss*, the combination of losses from a base-model and sub-models. This loss allows the base model to be able to detached into sub-models during inference and reduces overall complexity to the user comparing with NSN.

By detach convolutional layers on demand, CNSN promises the greater reduction in MAC than NSN which can detach only fully connected layers. Our main contributions in this research are listed as follows:

- We propose CNSN, a CNN with ability to detach a base-model into sub models.
- We introduce *multi-model loss*, a combination of losses from base-model and sub-models.
- We introduce a *step-down convolutional layer* that acts as the input layer to sub-models.

## 2. Convolutional Network with Sub-Networks

CNSN consists of a base-model and several sub-models. The sub-model is a subset of the base-model with an exception, *step-down convolutional layer* that is not included in the base-model. Therefore, all parameters from sub-models except the *step-down convolutional layer* is identical to the base-model. In term of memory usage, we only required to store the base-model and several *step-down convolutional layers.*

Overview of CNSN is illustrated in Fig. 1. Fig. 1 illustrates a base-model at the top row and two sub-models at the two bottom rows. Each base- and sub-models can directly receive the input image and independently product the prediction to other models. With less amount of parameters in the model, the sub-model has less capacity comparing to the base-model. However, with the same reason, it promises to perform the inference faster than the base-model.

With both base- and sub-models, CNSN consists of the three components to make CNSN operate-able with CNN. These three components are as follows: *step-down convolutional layer*, *sub-models*, and *multi-losses*.

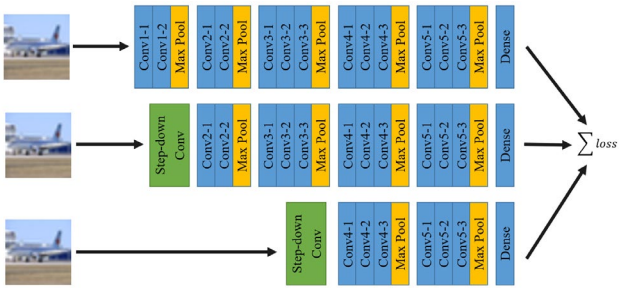


Fig. 1. Overview of CNSN. The top row represents a base model of CNSN while two bottom rows represent the sub-models. Each model can receive the image and produce the prediction. During training, losses from all model are collected as *multi-model loss.*

### 2.1. *Step-down Convolutional Layer*

The *step-down convolutional layer* is a convolutional layer that is designed to solve the spatial difference between the input and hidden layer in the CNSN. By solving this problem, this allows CNSN to reuse a hidden convolutional layer of base model as an input layer of sub model. However, to achieve this, there are some concerns as follows.

The first problem is the number of input filters of hidden layer may not be a same comparing to the input layer. With this issue, the hidden layer cannot process the input image. To solve this problem, we can place the *step-down convolutional layer* with the same amount of input filters to the input layer. Another issue is the expected size of the input feature is difference between base- to sub-models. By adjusting the number of stride and size of kernel of *step-down convolutional layer*, we can adjust the shape of output feature maps to be process further in the sub-model.

### 2.2. *Sub-models*

Instead of allowing every layer to be detach-able as in NSN, CNSN only allows few sub-models to be detached. In each batch trainings, both NSN and CNSN requires base- and sub-models to be propagated in the same iteration. This causes the problem that is parameters from base- and sub-models may not be fitted in a single GPU if the base-model is large and the base-model contains many sub-models. Therefore, to able to train on the bigger CNN with CNSN, we reduce the number of detach ability to few sub-models instead.

### 2.3. *Multi-model loss*

To allow both base-model and sub-models to operate as an individual model, we purpose a method called *multi-*

*model loss*. The *multi-model loss* is designed to balance between the loss from base-model and sub-models. By assume there are *N* sub-models in the base-model, the *multi-model loss* or $l_{all}$ is formulated as shown in Eq. (1). Where $l_N$ is the loss from a *N* sub-model, and $l_{base}$ is a loss from the base-model. To utilize this loss, all models must be forward propagated in the same batch training first to find $l_N$ and $l_{base}$. Then, we can use $l_N$ and $l_{base}$ to find $l_{all}$.

$$l_{all} = \sum_{1}^{N} l_N + l_{base} \quad (1)$$

## 3. Experimental results and Discussion

We implemented our CNSN using VGG-16[8]-like model as the base-model. Since VGG-16 is designed for ImageNet Large Scale Visual Recognition Challenge[9] (ILSVRC) 2014, some layers in VGG-16 are required to be modified to able to operate with images from CIFAR10[10]. We removed first two fully-connected layers and modified the last fully-connected layer to contain 512 input neurons and ten output neurons. To stabilize the training process, we also attach a batch normalization[11] (BN) layers after each weight layers except on the last fully-connected layer. We utilized ReLU as an activation function except the last fully-connected layer that we assigned with the log-softmax instead.

In this experiment, the overall setting is illustrated in Fig. 1. We assigned with two sub-models within the base-model. Without including any of activation, BN, and pooling layer, the first sub-model or *sub-model0* is the base-model after removing first two convolutional layers. The second sub-model, *sub-model1* is the base-model after removing first seven convolutional layers. The *step-down convolutional layer* for *sub-model0* is the convolutional layer with stride two and kernel size two. For *sub-model1*, it is the convolutional layer with stride eight and kernel size eight. We defined the *baseline model* as a VGG-16-like model that was trained without any modifications.

We conducted a benchmark with CIFAR10 dataset. CIFAR10 consists of 10 different classes. Each class consists of 5,000 training and 1,000 test images. We performed the data augmentations by padding 4 pixels into the training image and randomly crop back to original size. The training images were further augmented by horizontal flipping and normalizing with a channel-wise mean and standard deviation of CIFAR10 dataset. We trained all models with stochastic gradient descent with momentum of 0.9. We set an initial learning rate as $10^{-2}$ and step-down to one-tenth after we trained for 50, 100 and 150 epoch. We warmed-up the learning rate for an epoch and trained for totally 300 epochs using the training batch size as 32. We reported the best test accuracy that occurred during the training.

The experimental results are as shown as Table 1. In Table 1, we compared the CNSN base model with the *baseline model* with the same setting as CNSN except for the weight decay. We found out the optimized weight decays are differed between the *baseline model.* and CNSN. Therefore, we applied the different weight decay to each model.

Table 1. Results of CNSN models comparing a base-line model on CIFAR10 dataset.

| | Test accuracy | MAC | Weight decay |
|---|---|---|---|
| *sub-model0* | 0.7738 | **0.124 G** | $6 \times 10^{-4}$ |
| *sub-model1* | 0.9115 | 0.275 G | |
| *base-model* | 0.9267 | 0.314 G | |
| *baseline model* | **0.9316** | 0.314 G | $6 \times 10^{-3}$ |

Our *base-model* achieved the loss in test accuracy for 0.0049 in exchanging to the ability to detach into sub-models. The *sub-models1* able to reduces more than half of MAC comparing with the *baseline model*, however this came with the significantly drop in term of test accuracy.

## 4. Conclusion

We propose CNSN, a CNN that can be detached into smaller CNNs on fly. To gain the detach ability to CNN, we propose the *multi-model loss* and *step-down convolutional layer.* The base-model of CNSN can deliver the performance that is compare-able with the regular trained models, while sub-models significantly reduces the amount of MAC, however with the trade-off in loss in test accuracy.

**Acknowledgements**
This research was supported by JSPS KAKENHI Grant Numbers 17K20010.

## References

1. Tan, Mingxing, and Q. V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *arXiv preprint arXiv:1905.11946* (2019).
2. Chen, Liang-Chieh, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation." In *Proceedings of the European conference on computer vision (ECCV)*, pp. 801-818. 2018.

3. Tan, Mingxing, Ruoming Pang, and Quoc V. Le. "Efficientdet: Scalable and efficient object detection." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10781-10790. 2020.
4. Li, Hao, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning filters for efficient convnets." *arXiv preprint arXiv:1608.08710*, 2016.
5. Yu, Jiahui, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. "Slimmable neural networks." *arXiv preprint arXiv:1812.08928*, 2018.
6. Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Branchynet: Fast inference via early exiting from deep neural networks." In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464-2469. IEEE, 2016.
7. Fuengfusin, Ninnart, and Hakaru Tamukoh. "Network with Sub-Networks." Proceedings of International Conference on Artificial Life and Robotics, vol. 25, 2020, pp. 191–194.
8. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556*, 2014.
9. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C., 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), pp.211-252.
10. Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "The cifar-10 dataset." *online: http://www.cs.toronto. edu/kriz/cifar. html* 55 (2014).
11. Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167*, 2015.