WeNet: Configurable Neural Network with Dynamic Weight-Enabling for Efficient Inference

Jingxiao Ma
School of Engineering
Brown University
Providence, Rhode Island, RI
jingxiao_ma@brown.edu

Sherief Reda
School of Engineering
Brown University
Providence, Rhode Island, RI
sherief_reda@brown.edu

2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) | 979-8-3503-1175-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISLPED58423.2023.10244723

***Abstract*—Deep Neural Networks (DNN) are widely deployed in resource-limited edge devices. Due to the limitation of computational resources, it is important to meet the timing and energy constraints while maintaining a high level of accuracy. To deploy the same DNN model on different edge devices, one challenge is to train a dynamic neural network with the flexibility of balancing the trade-off between accuracy and efficiency at runtime. In this paper, we present a novel methodology, dynamic Weight-enabling Network (WeNet), where the weights of neural network can be dynamically enabled or disabled to switch between different sub-networks, so that we are able to balance the trade-off between inference time, energy consumption and model accuracy. We extend the methodology to convolutional layers using group convolution and channel shuffling. We also propose a design space exploration approach to search for the optimal sub-network for different scenarios. We thoroughly evaluate our methodology using a number of DNN architectures on different hardware platforms, showing that WeNet provides a large number of energy-efficient operation modes, 73.2% of which provide better accuracy-efficiency trade-off compared to other methodologies.**

***Index Terms*—Deep learning, Energy-efficient application, Dynamic neural network.**

I. Introduction

Deep Neural Networks (DNN) are widely used in many applications, *e.g.* object detection, gesture recognition and augmented reality, which are often deployed on edge devices with limited computational resources. While it is possible to train DNNs on the cloud, inference on edge devices needs to meet timing and energy constraints. Different devices have different computing power, while other running tasks also limit the amount of available computational resources. To deploy the same DNN model in different scenarios, one solution is to train a *dynamic DNN* with *multiple configurations of operation modes*, where we are able to choose the most applicable configuration at runtime to meet the timing and energy constraints while optimizing accuracy.

The last few years have seen various methodologies for dynamic DNNs, which can be categorized into three classes: flexible width [1]–[3], flexible depth [4], [5] and flexible precision [6], [7]. In this paper, we propose a novel orientation, *flexible weight-enabling*, where connective weights between layers can be dynamically enabled or disabled. As Figure 1 illustrates, the entire model is first trained on server. For inference, the model can be dynamically configured to different sub-networks by enabling different set of weights to meet the timing and energy constraints of different devices. We name our methodology *dynamic Weight-enabling Network* (WeNet). The contributions of this paper are as follow.

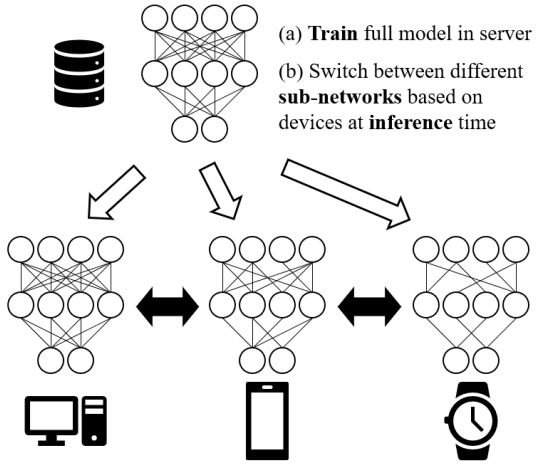


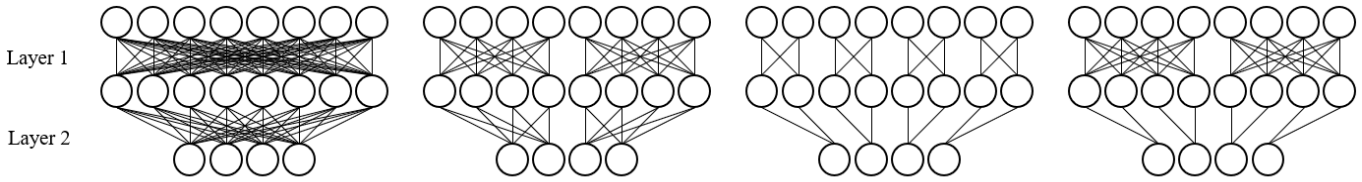
Fig. 1. Flexible weight-enabling methodology, where weights can be dynamically enabled to form different sub-networks for different hardware platforms.

- We introduce a novel dynamic DNN architecture, WeNet, that is able to dynamically enable different subsets of weights at runtime. We propose a weight-enabling pattern, that for each layer, the subset of weights forms a sub-network with several *independent groups*.
- We extend WeNet to convolutional layers by using *flexible group convolution* and *channel shuffling operation*.
- During training, random sub-networks are sampled at each iteration, where *switchable batch normalization* [2] is used. At inference time, we propose a *design space exploration* method to search optimal sub-networks and balance the trade-off between efficiency and accuracy.
- We evaluate WeNet using multiple DNN architectures, and measure inference time, energy consumption and accuracy with different configurations of sub-networks. By comparing against other dynamic DNNs, we demonstrate that WeNet provides better accuracy-efficiency trade-off.

The organization of this paper is as follow. In section II,

979-8-3503-1175-4/23/$31.00 ©2023 IEEE

Authorized licensed use limited to: Brown University. Downloaded on August 27,2025 at 00:38:50 UTC from IEEE Xplore. Restrictions apply.



(a) Full network (1 channel) (b) 1/2-network (2 channels) (c) 1/4-network (4 channels) (d) Mixture of 1/2 and 1/4 networks

Fig. 2. Example of a WeNet on dense layers. (a) Full network: Enable all weights to restore original network with highest accuracy. (b) $1/2$-network: Enable $1/2$ weights and form 2 separated channels. (c) $1/4$-network: Enable $1/4$ weights and form 4 separated channels. (d) Combination of $1/2$- and $1/4$-network.

TABLE I
COMPARISON BETWEEN DIFFERENT DYNAMIC NETWORK METHODS

	Memory Footprint	Num. of Neurons	Special Device
Flex. Width	Reduced	Reduced	Not Needed
Flex. Depth	May Increased	Reduced	Not Needed
Flex. Precision	Reduced	Same	Needed
Flex. W.E.	Reduced	Same	Not Needed

we overview relevant previous works. In section III, we introduce WeNet and its training algorithms. We provide our experimental results in section IV. Finally, we summarize our conclusion and directions for future works in Section V.

II. PREVIOUS WORK

A number of methodologies for efficient DNNs have been proposed [8]–[10]. While these methodologies aim to shrink model size and reduce computational cost, none of them have the flexibility of adjusting model efficiency at runtime. The first design with such flexibility is "Big/Little" implementation [11], where two networks are trained and the "big" network is triggered only if the result from "small" network is not deemed confident enough. However, such methodology needs to store multiple models, and the latency for switching between different models is not negligible. Thus, a more efficient approach is to train *one single* network, with the flexibility to switch between different modes at runtime. Recent works on dynamic network can be categorized into three classes:

- **Flexible Width:** At runtime, the width of each layer can be shrunk to reduce computational cost, or expanded to increase accuracy. Tann *et al.* [1] propose one of the first runtime configurable DNN with flexible width, where DNN is trained incrementally at each width ratio. Slimmable neural network (SNN) [2] further improves the idea with switchable batch normalization. US-Nets [3] extends SNN to execute at arbitrary width ratio.
- **Flexible Depth:** Flexible depth means that the depth of DNN is adjustable at runtime. One common approach is to introduce early-exiting points, where the rest layers are ignored [5]. BranchyNet [4] proposes to add side branches to exiting points, where a forward pass can exit earlier from the main branch with higher confident inputs.
- **Flexible Precision:** Flexible precision means that the precision of operands, including both weights and inputs, can be adjusted dynamically at runtime. Pagliari *et al.* [6] find that many inputs do not need full precision to make accurate classification, and propose to reduce precision of operations when the confidence of input is high enough. SP-Nets [7] propose to train a multi-precision network using switchable batch normalization.

Table I compares three categories of dynamic networks, where all of them aim to reduce inference time and energy consumption. For *flexible width*, the number of neurons activated at each layer can be reduced at runtime, which leads to less memory footprint. For *flexible depth*, however, due to the additional side branches, memory footprint may even increase. Both *flexible width* and *flexible depth* reduce number of neurons, where crucial information may be lost and accuracy may drop significantly. For *flexible precision*, each weight uses less memory space by decreasing its floating-point precision, which reduces memory footprint. Since the number of neurons remains the same as original network, it is more likely to retain extracted features and remain high accuracy. But this class of dynamic networks cannot be used on any arbitrary devices, since the device has to support operations with multiple precision to exploit the efficiency of low-precision operands. Comparing to all three categories, we expect our methodology to reduce memory footprint for energy-efficient inference, keep the number of neurons unchanged for higher accuracy, while generalizing to all types of devices. Thus, as Table I shows, we propose *flexible weight-enabling* (Flex. W.E.), where different subsets of weights can be enabled dynamically at runtime.

III. PROPOSED METHODOLOGY

In this section, we propose the methodology of Weight-enabling Network (WeNet). As Figure 2 shows, WeNet enables different subsets of weights to dynamically switch between different sub-networks, where computational cost can be adjusted without changing the number of activated neurons. Such network architecture can be executed on any types of hardware platforms, though as discussed later, it will benefit even more with parallelism. After discussing WeNet and its extension to convolutional layers, we will introduce training algorithm and design space exploration method for optimizing the trade-off between efficiency and accuracy at inference time.

A. Dynamic Weight-enabling Network (WeNet)

WeNet dynamically balances between accuracy and efficiency by switching between different sub-networks, each of which enables a subset of weights, following a specific pattern as shown in Figure 2. Assume for a dense layer, input vector is $\mathbf{x}$ and output vector is $\mathbf{y}$. In a standard dense layer, every neuron in $\mathbf{y}$ is connected to every neuron in $\mathbf{x}$, which leads to $\ell(\mathbf{x}) \cdot \ell(\mathbf{y})$ weights, where $\ell(\mathbf{x})$ denotes the number of neurons in $\mathbf{x}$. In WeNet, we propose to divide both input $\mathbf{x}$ and output $\mathbf{y}$

Authorized licensed use limited to: Brown University. Downloaded on August 27,2025 at 00:38:50 UTC from IEEE Xplore. Restrictions apply.

into n groups, respectively, such that $\mathbf{x} = \mathbf{x}_1 \cup \mathbf{x}_2 \cup \ldots \cup \mathbf{x}_n$ and $\mathbf{y} = \mathbf{y}_1 \cup \mathbf{y}_2 \cup \ldots \cup \mathbf{y}_n$. To reduce the computational cost, instead of connecting all input and output neurons, only neurons in $\mathbf{y}_i$ are connected to neurons in $\mathbf{x}_i$, which forms a partially-connected sub-network. In other word, for each layer, $(\mathbf{x}_i, \mathbf{y}_i)$ pairs form *independent groups*. By adjusting the number of groups n at runtime, WeNet dynamically controls the number of enabled weights, and thus adjusts computational cost.

Notice that all *independent groups* can be executed in parallel. To optimize inference time and energy consumption, each layer is evenly divided, in other word, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ have same number of neurons. Evenly-divided layers have two benefits: (1) For layers with n groups, evenly division leads to the least number of weights as well as computational cost

$$O\left(\frac{\ell(\mathbf{x}) \cdot \ell(\mathbf{y})}{n}\right) \tag{1}$$

(2) Each group has the same amount of FLOPs, which further improves efficiency with parallel execution.

Figure 2(a) shows the full network. To improve efficiency, we may disable half of the weights and execute the $1/2$-network as Figure 2(b) shows. If faster inference time or lower energy consumption is expected, we may then switch to $1/4$-network shown in figure 2(c). As less weights are enabled during this process, WeNet loses more information, which leads to lower accuracy as trade-off. To further improve the flexibility, we may create more operating modes, where each layer may have individual weight-enabling ratio. Figure 2(d) shows a combination of $1/2$- and $1/4$-network. By enabling different subset of weights, WeNet dynamically switches between sub-networks to balance between efficiency and accuracy.

B. WeNet on Convolutional Layers

WeNet transforms standard dense layers into a set of switchable partially-connected sub-networks. Nowadays, the most widely used layers in DNNs are convolutional layers, which itself is a partially-connected layer. In this subsection, we extend the idea of WeNet to convolutional layers.

Similar to enabling subset of weights in dense layers as shown in Figure 2, in convolutional layer, WeNet dynamically enable *subset of kernels* between feature maps, which forms *independent groups of channels*, or in other word, using group convolution with the flexibility of switching number of groups. We consider a standard convolutional layer as the full network of convolutional WeNet. Assume that it takes an input tensor L_i of size $h_i \times w_i \times c_i$, and applies convolutional kernel $K \in \mathbb{R}^{k \times k \times c_i \times c_o}$ to produce an output tensor L_o of size $h_o \times w_o \times c_o$. The computational cost of standard convolution is

$$T_{full} = h_i \times w_i \times c_i \times c_o \times k \times k \tag{2}$$

Similar to WeNet on dense layer, input tensor L_i with c_i feature maps is divided *evenly* into n groups, each of which has c_i/n feature maps, such that $L_i = L_i^1 \cup L_i^2 \cup \ldots \cup L_i^n$. Output tensor L_o is also divided in same pattern $L_o = L_o^1 \cup L_o^2 \cup \ldots \cup L_o^n$. In standard convolution, there exist kernels that convolve any input feature map into any output feature map. In sub-network of WeNet, however, only kernels between

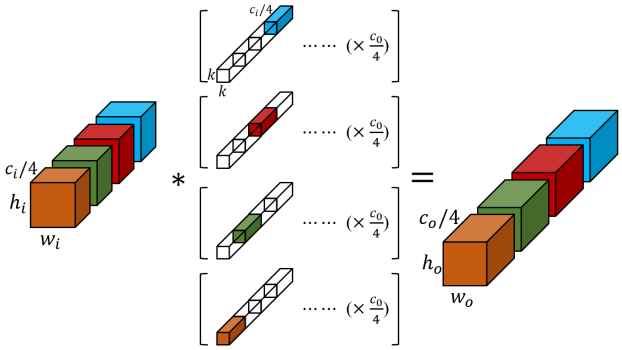


Fig. 3. $1/4$-network of convolutional layer

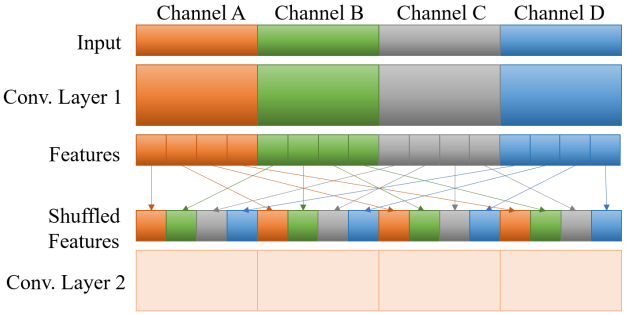


Fig. 4. Channel shuffling after $1/4$-network

feature maps of L_i^j and L_o^j is enabled. Figure 3 demonstrates the example of dividing input and output tensors into 4 groups, where input feature maps in L_i^j are only convolved to output feature maps in L_o^j, using only the kernels shown in same color. Since WeNet effectively forms 4 *separate groups of channels*, only $1/4$ of kernels in each filter are enabled. Thus, computation cost is reduced to $1/4$ of original cost T_{full}.

$$T_{1/4} = h_i \times w_i \times \frac{c_i \times c_o \times k \times k}{4} = \frac{T_{full}}{4} \tag{3}$$

Similar to dense layer, each group of channels is independent, which can be executed in parallel to further improve efficiency at inference time. However, on the other hand, each feature map can only receive inputs from its own group. Compared to standard convolution, feature maps in each group receive limited input information for features extractions, where accuracy may drop significantly. To mitigate the accuracy loss caused by *isolated* groups, we adopt a channel-shuffling operation proposed by ShuffleNet [9]. A channel-shuffling operation is performed between two convolutional layers. As Figure 4 shows, after the first convolutional layer, each group of features is first split, and then shuffled and merged to form new groups of features as input to the next convolutional layer. Although a standard convolution is replaced by several independent groups of channels, channel-shuffling allows each group to learn from the others, which significantly improves accuracy. Meanwhile, parallelism can still be used on each group to improve efficiency.

Implementation of channel-shuffling is straightforward. Suppose a convolutional layer of WeNet has *maximum* g independent groups (in experiment, we set $g = 16$), where each group has n feature maps. All $g \times n$ feature maps are

Authorized licensed use limited to: Brown University. Downloaded on August 27,2025 at 00:38:50 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Train with random sampling and S-BN

Input : Training set $(\mathbf{X}, \mathbf{Y})$ with features $\mathbf{X}$ and labels $\mathbf{Y}$, Number of iterations n_{iter}, Number of sampled sub-network per iteration s, Loss function $loss\_fn$, Optimizer opt
Output: WeNet M

1 Initialize WeNet M
2 Initialize S-BN layers for each weight-enabling pattern
3 **for** $num\_iter = 1, 2, \ldots, n_{iter}$ **do**
4 Get next batch of data and labels $(\mathbf{x}, \mathbf{y})$ from $(\mathbf{X}, \mathbf{Y})$
5 Clear gradients $opt.zero\_grad()$
6 Execute full network $\hat{y} = M(x)$
7 Compute loss $loss = loss\_fn(\hat{y}, y)$
8 Accumulate gradient $loss.backward()$
9 Adjust S-BN to smallest sub-network
10 Execute smallest sub-network $\hat{y} = M'(x)$
11 Compute loss $loss = loss\_fn(\hat{y}, y)$
12 Accumulate gradient $loss.backward()$
13 Randomly sample $s - 2$ sub-networks
14 **for** *each sampled WeNET* **do**
15 Adjust S-BN according to weight-enabling patterns
16 Forward training data $\hat{y} = M'(x)$
17 Compute loss $loss = loss\_fn(\hat{y}, y)$
18 Accumulate gradient $loss.backward()$
19 **end**
20 Update gradients to weight $opt.step()$
21 **end**
22 **return** M

first reshaped into (g, n), which are then transposed to (n, g) and flattened as shuffled groups. Channel-shuffling operation only involves matrix transposing, where the computational cost is negligible compared to model inference.

C. Training WeNet with Switchable Batch Normalization

WeNet dynamically enables subset of weights or kernels to form sub-networks, where each layer may have different number of *independent groups*. Since it is impractical to enumerate and train all possible sub-networks, we propose to *randomly sample* sub-networks at each iteration.

We also need to reconsider the implementation of Batch Normalization (BN) layers, which normalize feature maps across each batch. Nowadays, BN is widely used to reduce internal covariate and stabilize training process. Using y and y' to denote inputs and outputs, computation of BN layers is defined as

$$y' = \gamma \cdot \frac{y - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{4}$$

where μ, σ^2 are means and variance of feature maps of current batch, γ, β are learnable variables. Since each layer has a number of possible weight-enabling pattern, where each neuron receives different inputs as Figure 2, μ and σ^2 are not consistent, which leads to inaccurate learning of γ and β. Yu *et al.* propose Switchable Batch Normalization (S-BN) in SNN [2], which privatizes μ, σ^2, γ, β settings for different pattern of the same layer. In other words, each weight-enabling pattern has its own customized BN layer, which is switchable according to the selected pattern. The number of learnable variables in BN layers is negligible compared to other weights and kernels. Notice that in WeNet, although there is a large number of possible sub-networks, for each layer the possible number of weight-enabling patterns is limited (in our experiment, maximum 16 independent groups). Thus, it is still efficient to use customized S-BN layers for all weight-enabling patterns in each layer. Algorithm 1 describes the process of training WeNet using S-BN. We initialize DNN with independent BN layers for each weight-enabling pattern (line 1-2). For each iteration, we accumulate gradients from full network (line 6-8), the smallest sub-network (line 9-12) and other $s - 2$ random sub-networks (line 14-19).

Algorithm 2: Design space exploration of WeNet

Input : Full network of WeNet M, Accuracy threshold δ
Output: Optimal sub-network M'

1 $M' = M$ // Keep track of most efficient sub-network so far
2 $acc =$ Accuracy of M // Accuracy of current sub-network
3 **while** $acc > \delta$ **do**
4 **for** *each layer* i *in* M' *that has* < 16 *groups* **do**
5 Increase the number of groups in layer i as sub-network m_i
6 $acc_i =$ Accuracy of m_i
7 $t_i =$ Inference time of m_i
8 $loss_i = t_i / acc_i$
9 **end**
10 $k = \arg\min_i loss_i$
11 $M' = m_k$ and $acc = acc_k$
12 **end**
13 **return** M'

D. Design Space Exploration

After training with Algorithm 1, theoretically WeNet can be configured into operation modes corresponding to all possible sub-networks. But in practice, some sub-networks performs better in terms of efficiency-accuracy trade-off. Given an accuracy threshold, to determine the optimal operation mode on a specific hardware platform, we propose to explore the design space of possible sub-networks, using *design space exploration* (DSE) approach as Algorithm 2.

We start from the full network (line 1-2). In each iteration of DSE, we consider a number of candidate sub-networks by increasing the number of groups in one of the WeNet layers, respectively (line 5). We set an upper bound on number of groups (16 in experiment) to prevent accuracy dropping rapidly. As our objective is to minimizing computational cost (represented by inference time t) while keeping accuracy acc high, for each candidate sub-network i, we evaluate t_i and acc_i using targeted hardware device, and compute a loss using t_i / acc_i (line 8). By minimizing the loss in each iteration, we keep reducing computational cost while optimizing the trade-off against accuracy. When accuracy of current sub-network drops below threshold, DSE is finished.

Using DSE, we effectively explore a number of possible sub-networks. At each iteration, sub-network with smallest loss forms Pareto Frontier, which demonstrate the optimal trade-off between efficiency and accuracy.

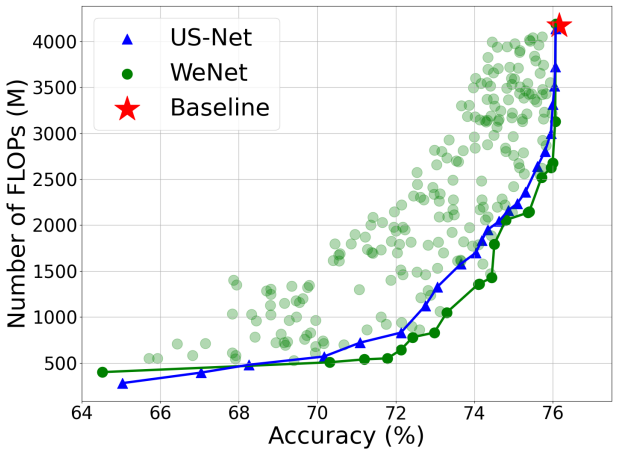
IV. Experimental Results

In this section, we first describe our experiment setup. To highlight the benefits of channel-shuffling, we evaluate WeNet

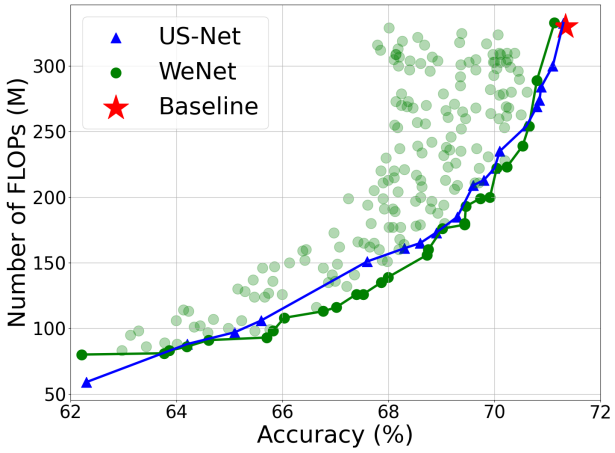
Authorized licensed use limited to: Brown University. Downloaded on August 27,2025 at 00:38:50 UTC from IEEE Xplore. Restrictions apply.

TABLE II
COMPARISON OF TOP-1 ACCURACY AND INFERENCE TIME WITH AND WITHOUT CHANNEL-SHUFFLING OPERATIONS USING RESNET-50.

Sub-Network	Without Shuffling		With Shuffling	
	Acc.(%)	Time(ms)	Acc.(%)	Time(ms)
1/16	60.88	41.68	64.24(+3.36)	43.41(+1.73)
1/8	65.81	82.79	70.43(+4.62)	85.62(+2.38)
1/4	69.38	124.14	74.07(+4.69)	127.35(+3.21)
1/2	73.49	127.88	75.63(+2.41)	132.17(+4.29)
Full	75.99	198.62	76.07(+0.08)	203.38(+4.76)
Average	69.11	115.02	72.09(+2.98)	118.38(+3.36)



(a) ResNet-50



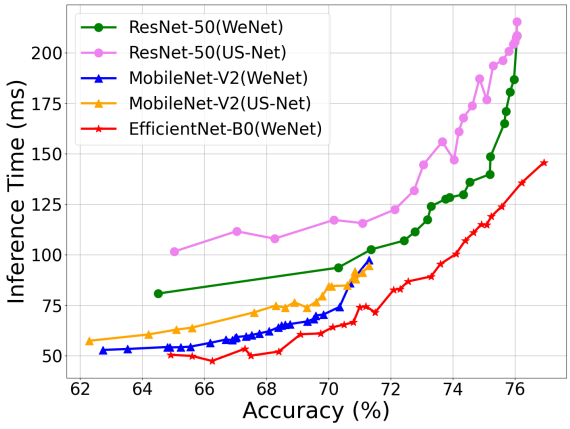
(b) MobileNet-V2

Fig. 5. Comparison between WeNet and US-Net [3]

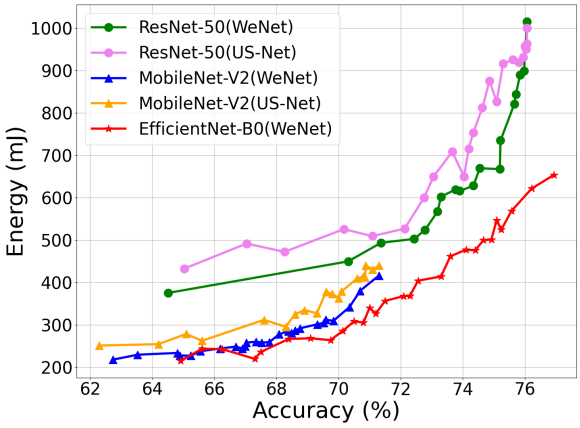
with and without channel-shuffling operations. We also explore the design space of WeNet and compare optimal execution modes against other methodologies, showing that WeNet provides better trade-off. Finally, we measure inference time and energy consumption, showing the benefits on different types of real devices.

A. Experiment Setup

1) *Datasets and models:* We implement and evaluate WeNet using *ResNet-50* [12], *MobileNet-V2* [8] and *EfficientNet-B0* [10] on ImageNet classification problem.
2) *Software setup:* We implement weight-enabling operations with *group convolution* in *PyTorch*. We use default training settings for each benchmark, with one additional hyper-parameter $s = 20$, meaning that in each iteration we randomly sample 20 sub-networks.
3) *Hardware setup:* We measure inference time and energy consumption on the NVIDIA Jetson Nano board.



(a) Inference time



(b) Energy consumption

Fig. 6. Inference Time and Energy Consumption on Jetson Nano Board

B. Channel-Shuffling

In the first experiment, we demonstrate the benefit of channel-shuffling operation by evaluating five operating modes of WeNet with and without channel-shuffling. Table II shows the comparison result using ResNet-50, where each residual block has one channel-shuffling operation after the first 1×1 convolution layer. Channel-shuffling operation significantly improves accuracy for all five sub-networks. On average, channel-shuffling increases accuracy by 2.98%. Meanwhile, channel-shuffling introduces additional computational cost. On average, these operations use additional $3.36ms$ out of $118.38ms$ inference time. Compared to significant improvements on accuracy, increasing of inference time is negligible.

C. WeNet v.s. US-Net

In the second set of experiment, we compare WeNet against US-Net [3], which provides multiple operation modes by adjusting width ratio for each layer, using two benchmarks, i.e. *ResNet-50* [12] and *MobileNet-V2* [8]. After training WeNet

Authorized licensed use limited to: Brown University. Downloaded on August 27,2025 at 00:38:50 UTC from IEEE Xplore. Restrictions apply.

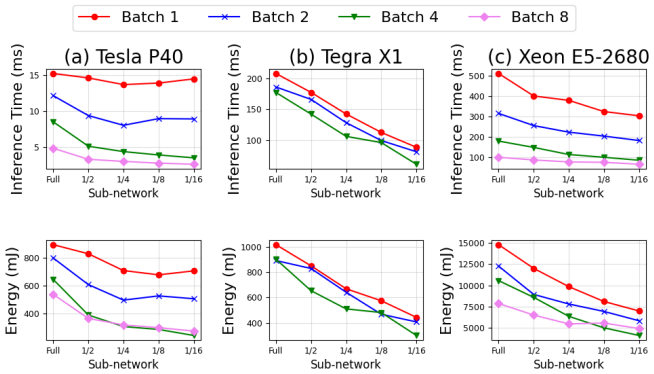


Fig. 7. Evaluation of *ResNet50* on three devices, with different batch size

using Algorithm 1, we explore the design space of WeNet operation modes using Algorithm 2, where computational cost is measured by number of Floating-Point Operations (FLOPs). Figure 5 shows the comparison results, where red star represents the original model. Compared to US-Nets, WeNet has less number of FLOPs in most accuracy range, which demonstrates the state-of-the-art performance. For ResNet-50, WeNet provides more efficient operating modes when accuracy is higher than 69%. For MobileNet-V2, WeNet performs better when accuracy is between 64% and 71%.

D. Inference Time and Energy Consumption

In the third set of experiment, we implement WeNet using all three benchmarks and plot operation modes on Pareto Frontier. Using Jetson Nano board, we evaluate inference time and energy consumption of these optimal points. Figure 6 shows the evaluation results, including US-Net in Section IV-C as comparison. As Figure 6a shows, for all three benchmarks, WeNet substantially saves inference time by trading-off small amount of accuracy. Compare to number of FLOPs in Figure 5, the improvement of inference time is more significant, which is always lower then US-Net. Since each layer of WeNet consists of *independent* groups, which can be executed in parallel, inference time can be further saved if there are idle threads (or warps). Thus, the benefit of inference time is more significant. As Figure 6b shows, WeNet also substantially improves energy consumption compared against US-Net.

E. Evaluation on Different Devices

Finally, in Figure 7, we measure inference time of models trained in Section IV-C using three devices, *i.e. Tesla P40* (GPU), *Tegra X1* (GPU) and *Xeon E5-2680* (CPU). High-performance GPUs (*Tesla P40*) are already well-optimized for tensor operations. Thus, on these devices, inference time of *full network* is already fast enough. Disabling part of weights does not accelerate inference time as expected, but does improve energy efficiency due to fewer FLOPs. If we increase batch size, inference time and energy consumption keep reducing, since there is still enough computational resources to do parallel computing for each batch. On the other hand, for low-performance devices (*Tegra X1*), all threads (or warps) are busy even if batch size is 1. In this case, increasing batch size does not make much difference. But inference time and energy consumption vary a lot according to sub-network, where disabling more weights improves efficiency a lot. Since CPU (*Xeon E5-2680*) is not designed specifically for tensor operations, the inference time and energy consumption is much higher compared to GPU. But disabling more weights still improves energy efficiency significantly.

V. Conclusion

In this paper, we proposed a novel dynamic network methodology, WeNet, which enables different subsets of weights on the fly to trade-off between accuracy and inference time. By enabling smaller subsets of weights, WeNet effectively forms a "sparser" sub-network with multiple separate groups of channels, where parallelism can be used to further improve efficiency. We also extended WeNet to convolutional layers using group convolution and channel shuffling, trained with switchable batch normalization and explored design space of possible sub-networks. By evaluating on different DNNs, we demonstrated that WeNet is able to optimize the trade-off between efficiency and accuracy.

References

[1] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2016, pp. 1–10.

[2] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *arXiv preprint arXiv:1812.08928*, 2018.

[3] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.

[4] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[5] S. Laskaridis, A. Kouris, and N. D. Lane, "Adaptive inference through early-exit networks: Design, challenges and directions," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 2021, pp. 1–6.

[6] D. J. Pagliari, E. Macii, and M. Poncino, "Dynamic bit-width reconfiguration for energy-efficient deep learning hardware," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[7] L. Guerra, B. Zhuang, I. Reid, and T. Drummond, "Switchable precision neural networks," *arXiv preprint arXiv:2002.02815*, 2020.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[9] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[10] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[11] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *2015 international conference on hardware/software codesign and system synthesis (codes+ isss)*. IEEE, 2015, pp. 124–132.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

Authorized licensed use limited to: Brown University. Downloaded on August 27,2025 at 00:38:50 UTC from IEEE Xplore. Restrictions apply.