Fully Nested Neural Network for Adaptive Compression and Quantization

Yufei Cui[1] , **Ziquan Liu**[1] , **Wuguannan Yao**[2] , **Qiao Li**[1] ,
Antoni B. Chan[1] , **Tei-wei Kuo**[1,3] , **Chun Jason Xue**[1]
[1]Department of Computer Science, City University of Hong Kong
[2]Department of Mathematics, City University of Hong Kong
[3]Department of Computer Science & Information Engineering, National Taiwan University
{yufeicui3-c, ziquanliu2-c, satie.yao, qiaoli7-c}@my.cityu.edu.hk
{abchan, teiwei.kuo, jasonxue}@cityu.edu.hk

Abstract

Neural network compression and quantization are important tasks for fitting state-of-the-art models into the computational, memory and power constraints of mobile devices and embedded hardware. Recent approaches to model compression/quantization are based on reinforcement learning or search methods to compress/quantize the neural network for a specific hardware platform. However, these methods require multiple runs to compress/quantize the same base neural network to different hardware setups. In this work, we propose a fully nested neural network (FN^3) that runs only once to build a nested set of compressed/quantized models, which is optimal for different resource constraints. Specifically, we exploit the additive characteristic in different levels of building blocks in neural network and propose an ordered dropout (ODO) operation that ranks the building blocks. Given a trained FN^3, a fast heuristic search algorithm is run offline to find the optimal removal of components to maximize the accuracy under different constraints. Compared with the related works on adaptive neural network designed only for channels or bits, the proposed approach is unified for different levels of building blocks (bits, neurons, channels, residual paths and layers). Empirical results validate strong practical performance of the proposed approach.

1 Introduction

Deep neural networks have achieved great success in various domains, e.g., computer vision, natural language processing and etc [LeCun *et al.*, 2015]. To fit the computation-/memory-intensive neural network into mobile devices and embedded hardware, researchers either develop light-weight network architecture [Sandler *et al.*, 2018; Zhang *et al.*, 2018] or compress a pre-trained network through pruning [Han *et al.*, 2016; Neklyudov *et al.*, 2017]. To avoid the huge efforts on tuning hyper-parameter, network architecture search (NAS) is studied to automatically discover the best architecture with computation, memory or power constraints [Cai *et al.*, 2018; Jiang *et al.*, 2019]. Another method to fit the neural network into resource-limited devices is quantization, which quantizes full-precision weights into a small number of bits [Zhou *et al.*, 2016; Yang *et al.*, 2019]. As the optimal bit-widths for different layers might vary, reinforcement learning based automatic search methods are developed to find the optimal bit-width setups [Wang *et al.*, 2019].

The search-based methods alleviate the need to manually tune hyper-parameter in designing network architectures and to determine bit-width in quantization. However, the algorithms should be re-run once the resource budget (CPU clock rate, number of cores, size of RAM, etc.) changes. Recently, there is a rising attention on training one neural network that is adaptive to different resource budgets. These methods are particularly designed for either adaptive channel numbers [Yu *et al.*, 2019b] or adaptive quantization bits [Jin *et al.*, 2019]. As changing the channel numbers or bit widths leads to inconsistent batch normalization (BN) statistics, heuristics like maintaining multiple sets of switchable BN parameters are used, which leads to a quadratic number of BN parameters w.r.t. the number of channels/bits to be updated. [Yu and Huang, 2019] propose to solve this problem by re-collecting BN statistics after training, but the flexibility of the network is still limited (see Sec. 4).

The goal of this work is to provide a simple method to train a neural network once and yield optimal sub-networks (SN) for different budgets of resources. The SNs are *nested* in the sense that any smaller SN forms the basis of a larger one. In other words, whenever some building blocks are removed from a larger SN, the smaller SN comprising the remaining blocks is still complete and accurate. Moreover, the method is unified in that it can be applied to different levels of the network: layers, residual paths, channels, neurons and bits.

Our intuition of how to achieve the goal is from the classic boosting techniques [Zhou, 2012; Chen and Guestrin, 2016], where a strong classifier is formed by an expansion of weak classifiers. The weak classifiers are ordered such that the latter classifier approximates error between ground-truth and the prior classifier. From this perspective, boosting is nested in nature as a smaller expansion of classifier forms the basis of a larger one. In a well trained boosting model, removing the latter classifier does not affect the performance much. Thus, we exploit the similarity between several building blocks of a network and the weak classifier in boosting. We further

extend this inspiration to quantization bits by constructing nested quantization functions.

To rank the building blocks efficiently, we propose an ordered dropout (ODO) operation that samples a nested sub-network in each mini-batch during training. In contrast to standard dropout that acts on one block, ODO is applied to a sequence of building blocks to make them in order of importance. The intuition of ODO is to mimic the case that, during testing, an arbitrary number of blocks can be removed according to the resource budget. Mathematically, we show that, in expectation, each block maximizes a term of incremental information gain, thus the whole network is fully nested, which we call a *fully nested neural network* (FN^3). Note that boosting uses a greedy method to pick the next classifier, and does not update the previous classifiers. In contrast, ODO learns the whole set of nested classifiers simultaneously.

During the inference stage, to find the optimal curve of accuracy w.r.t. the number of blocks, we further propose a heuristic search algorithm, which leverages the order of blocks to reduce the search complexity from $\mathcal{O}(2^N)$ to $\mathcal{O}(N)$, where N is the number of blocks (see Sec. 3).

The experimental results show that FN^3-channel outperforms previous adaptive-channel methods, while FN^3 provides more sub-networks and consumes less training time. FN^3-bit also outperforms the adaptive-bit methods. We also show the performance of FN^3 on residual paths, for learning an adaptive architecture. An ablation study is conducted to show the effectiveness of the heuristic search algorithm.

2 Fully Nested Neural Network

The aim of this section is to find the basic building block of a nested neural network and a corresponding training scheme, such that during testing time, a certain number of blocks can be removed according to the resource budget, while maintaining high accuracy without re-training or fine-tuning.

Our inspiration comes from the family of *boosting* techniques, where $\mathbf{x}$ is the input data, a classifier $f_M(\mathbf{x})$ is composed with a sequence of basis functions $b_m = b(\mathbf{x};\gamma_m), m \in \{1,\dots,M\}$, and γ_m is the parameter of the m-th basis function. The *basis function expansions* are combined to produce the final prediction, $f_M(\mathbf{x}) = \sum_m \beta_m b(\mathbf{x};\gamma_m)$. Typically, these models are fit by minimizing this goal, $\min_{\{\beta_m,\gamma_m\}_1^M} \mathbb{E}_{\mathbf{x},y\sim p_{x,y}} L(y, \sum_{m=1}^M \beta_m b(\mathbf{x};\gamma_m))$. The additive characteristic of boosting allows us to remove some basis functions, while the remaining part of model still produce useful results.

Another characteristic of boosting [Hastie *et al.*, 2009] is that each basis function b_m approximates the residual under the current model, i.e., the difference between the ground-truth and the current prediction. For example, for squared-error loss,

$$L(y_i, f_{m-1}(\mathbf{x}) + \beta_m b(\mathbf{x}_i;\gamma_m)) = (r_{im} - \beta_m b(\mathbf{x}_i;\gamma_m))^2, \quad (1)$$

the m-th term $\beta_m b(\mathbf{x}_i;\gamma_m)$ best fits the residual error $r_{im} = y_i - f_{m-1}(\mathbf{x}_i)$ incurred by the previous expansion. Therefore, each b_m contributes less as m goes from 1 to M. The residual characteristic of boosting allows us to rank the basis functions according to the importance such that the less significant basis functions can be removed first. Once the additive basis functions are obtained and made to approximate the residual error, we can stack groups of basis functions to a deep model for better performance.

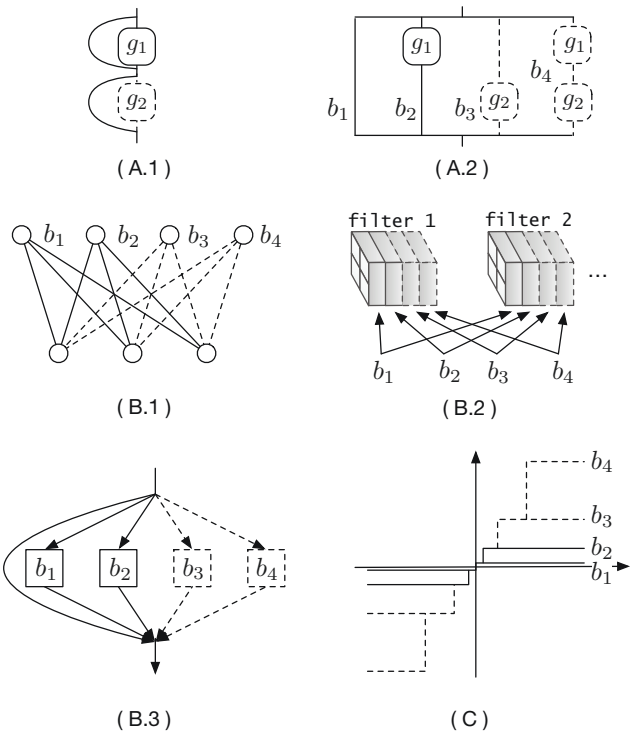

Figure 1: The additive building blocks of a NN with ODO, where $b_1 \dots b_4$ indicate the blocks. **Network-level**: (A.1) the output of a residual network as a sum of sub-networks, g_i is a layer; (A.2) unraveled view of the residual network, where b_m is composed of a sequence of g_i. **Layer-level**: (B.1) the output of a dense layer as a weighted sum of inputs; (B.2) a pixel in a feature map as a weighted sum of input channels; (B.3) a layer output as a sum of parallel residual blocks in ResNeXt. **Bit-level**: (C) quantization values as a sum of weighted bits. The dotted lines indicate dropped blocks (in this case the sampled index is $c = 2$) in one iteration. The solid lines indicate the remained blocks.

2.1 Building Blocks

Many parameterized building blocks[1] in a convolutional neural network (CNN) possess the additive characteristic (see Fig. 1). They can be classified into several categories:

Network-level: [Veit *et al.*, 2016] shows that a CNN with residual connections behaves like the ensemble of shallow networks. As seen in Fig. 1(A.1, A.2), the network output can be seen as the summation of $O(2^n)$ paths, $f_M(\mathbf{x}) = \sum_{m=1}^M b_m(\mathbf{x})$, $M = 2^n$, where each path is a combination of blocks from the set of n residual blocks, i.e., $b_m(\mathbf{x}) = g_n^{\mathbf{I}_n(m)}(\dots g_0^{\mathbf{I}_0(m)}(\mathbf{x}))$ and $\mathbf{I}_n(m)$ is the indicator function. We denote FN^3 with such block as FN^3-layer.

[1] In this paper, we use "blocks" and "basis functions" interchangeably.

Layer-level: 1) In a dense layer, as shown in Fig. 1(B.1), $\mathbf{h}_l = f(\mathbf{h}_{l-1}, \mathbf{\Gamma}) = \mathbf{\Gamma}^T \mathbf{h}_{l-1}$, where $\mathbf{h}_l$ is the l-th layer activation and $\mathbf{\Gamma}$ is the weight matrix. The i-th dimension of output can be written as the weighted sum of input, i.e., $\mathbf{h}_{l,i} = \mathbf{\Gamma}_i^T \mathbf{h}_{l-1}$. 2) In a convolutional layer, as shown in Fig. 1(B.2), every pixel on an output feature map can be written as the weighted sum of input channels, where the weights are from the channels of a convolutional filter. 3) In a residual block of ResNeXt, the output of a block can be written as sum of different paths, as shown in Fig. 1(B.3). We denote FN^3 with such blocks as FN^3-neuron/-channel/-path respectively.

Bit-level: An extreme case is to consider a quantization bit as a building block, as shown in Fig. 1(C). As directly training with integer weights is difficult, we instead use a nested quantization function $\bar{\mathcal{S}}(\cdot)$ which maps the full-precision weight w to $\bar{w}$. To construct $\bar{\mathcal{S}}(\cdot)$, we use the summation of several scaled and transformed Heaviside step functions, i.e., $\bar{w} = \bar{\mathcal{S}}(w_i) = \sum_j \alpha_j \mathcal{S}(w_i - z_j)$. Thus, the quantization function possesses the additive characteristic. We denote FN^3 with such block as FN^3-bit.

2.2 Ordered Dropout

Next we provide a unified method to rank the building blocks such that the blocks are nested. A naive approach is to imitate boosting: 1) train a few blocks as the base model; 2) fix the parameters of the base model, add new blocks to the model and train the new blocks; 3) repeat the second step until the total size of model meets the need. However, as the number of blocks varies among different layers, it is not obvious how to determine the number of augmented blocks (granularity) for different layers. Furthermore, the model would lack of flexibility if the granularity is large (e.g., slimmable NN [Yu *et al.*, 2019b]), while it would cause prohibitively long training time if the number is too small. Finally, since the previous blocks are fixed in each step, this naive training method would be not end-to-end trained, which could limit its accuracy.

We propose an operation named Ordered DropOut (ODO) that randomly selects nested sub-networks during training batches, which encourages ordering of the importance of each block. We label each block in a group with an index. ODO then samples a block index from a probabilistic distribution and drops all the blocks with indices greater than the chosen block. Specifically, a Categorical distribution $\mathcal{C}(m)$ is assigned over index $m \in \{1, \ldots, M\}$ of the blocks b_m. During training, we sample the index $c \sim \mathcal{C}(m)$ then disable the blocks with index $m > c$. The expansion with blocks dropped is written as $f_c(\mathbf{x}) = \sum_{i=1}^{c} \beta_i b(\mathbf{x}; \gamma_i)$, dubbed as *partial expansion*. The parameter of $\mathcal{C}(m)$ is set to be uniform, i.e., $p(m) = \frac{1}{M}$. As the m-th block is kept only when the sampled index $\leq m$, the probability of keeping m-th block is $\beta_m = \frac{M+1-m}{M}$. Since β_m is a monotonically decreasing function of m, then the blocks are forced to be ordered.

During testing, to make the output of a block equal to the expected output at training time, the output is also scaled by β_m. So the basis function expansion for a test data x is written as $f_M(x) = \sum_{m=1}^{M} \frac{M+1-m}{M} b(\mathbf{x}; \gamma_m)$.

The intuition of ordered dropout is to mimic the scenario that an arbitrary number of building blocks (bits, neurons, filters, paths, residual blocks etc.) can be removed during testing time. As the blocks with smaller indices are more likely to be kept, they obtain more information compared to higher-index blocks and thus become more important. Furthermore, as the higher-index blocks are disabled frequently, then the small-index blocks must be able to perform the task themselves. The problem of granularity is solved because the granularity is always as small as 1 block, while it does not increase the time/resource consumption. Fig. 1 shows how ODO can be applied to different blocks.

The idea of making dropout unit to have order was also explored by [Rippel *et al.*, 2014]. A "nested dropout" was applied to the hidden representation in an auto-encoder for unsupervised learning. The operation was proven to recover PCA under particular constraints. By contrast, we apply the ODO on building blocks of convolutional neural network for supervised learning, with the goal of adaptive deployment of neural network. We also prove that a NN trained with ODO and the MLE objective will lead to each block maximizing an incremental information gain over the previous blocks, yielding residual additive blocks (see next section).

2.3 Ordered Information

Next, we show that in a one-layer NN for a supervised learning task, training with ODO using the MLE objective leads to disentangled information gain between blocks.

We denote the ground truth joint likelihood as $\mathbf{x}, y \sim p_{\mathbf{x},y}$ and the discriminative model as $p_\Theta(y|\mathbf{x}) = p_0(y|f_{M,\Theta}(\mathbf{x}))$, where $\mathbf{x} \in \mathbb{R}^D, y \in \mathbb{R}$. Assume p_0 is probability of ground truth conditioned on the prediction. We define the prediction $\hat{y} = f_{M,\Theta}(\mathbf{x}) = \mathbf{v}^T \mathbf{z} = \mathbf{v}^T \sigma(\mathbf{U}^T \mathbf{x})$, where $\mathbf{z}$ is the hidden activation of a neural network and $\mathbf{U} \in \mathbb{R}^{D \times M}, \mathbf{v} \in \mathbb{R}^M$ are the parameters of the network. The full expansion can be written as $\hat{y} = f_{M,\Theta}(\mathbf{x}) = \sum_{i=1}^{M} \mathbf{v}_i \sigma(\mathbf{U}_i^T \mathbf{x})$, where $\mathbf{U}_i$ denotes the i-th column of matrix $\mathbf{U}$.

Proposition 1. *Under the above setting, maximizing the data log-likelihood is equivalent to maximizing the mutual information between ground-truth y and the prediction $\hat{y}$, i.e.,*

$$\max_\Theta \mathbb{E}_{\mathbf{x}, y \sim p_{\mathbf{x},y}} \log p_\Theta(y|\mathbf{x}) \Leftrightarrow \max_\Theta \mathbb{I}(y, \hat{y}). \tag{2}$$

where $\mathbb{I}(\cdot)$ is the mutual information[2].

Consider applying the proposed ODO on the elements of $\mathbf{v}$ (or equivalently on the columns of weight matrix $\mathbf{U}$), the problem becomes

$$\max_\Theta \mathbb{E}_{c \sim \mathcal{C}} \mathbb{I}(y, f_c(\mathbf{x})), \tag{3}$$

where $f_c(\mathbf{x}) = \sum_i^c b(\mathbf{x}; \mathbf{U}_i, \mathbf{v}_i) = \sum_i^c \mathbf{v}_i \sigma(\mathbf{U}_i^T \mathbf{x})$ is the partial expansion of basis functions.

Corollary 1. *The maximum likelihood objective is equivalent to*

$$\max_\Theta \mathbb{I}_1 + \frac{1}{M} \sum_{c=2}^{M} (M - c)(\mathbb{I}_c - \mathbb{I}_{c-1}), \tag{4}$$

where $\mathbb{I}_c = \mathbb{I}(y, f_c(\mathbf{x}))$ is the mutual information between the ground-truth and partial expansion[2].

[2]The proofs can be accessed via http://visal.cs.cityu.edu.hk/static/pubs/conf/ijcai20-fn3-sup.pdf.

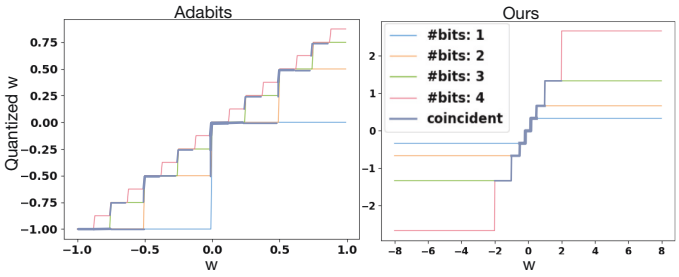

Figure 2: Comparison of quantization functions: (left) AdaBits; (right) our FN^3. The line thickness indicates the number of coincide curves.

Combining NN building blocks and ODO does not explicitly compose a Forward Stage-wise Additive Model, as in boosting in Eq. 1. However, Eq. 4 indicates the training objective is disentangled in a similar way. The objective includes a *base* mutual information $\mathbb{I}_1$ learned by the first block $b_1 = b(\cdot; \mathbf{U}_1, \mathbf{v}_1)$, as well as $\mathbb{I}_c - \mathbb{I}_{c-1}$, which is the *residual information* learned by block $b(\cdot; \mathbf{U}_c, \mathbf{v}_c)$. In other words, *in expectation, each building block maximizes the incremental information gain and aggregates the results for final prediction*. Thus, combining NN building blocks and ODO leads to *residual additive blocks*. This analysis can be easily extended to FN^3-layer, FN^3-path and FN^3-channel listed in Sec. 2.1.

2.4 Nested Quantized Neural Network

As shown in Sec. 2.1, the quantization function $\bar{\mathcal{S}}$ possesses the additive characteristic. ODO can be applied to the translated function $\mathcal{S}_j(w) = \mathcal{S}(w - z_j)$. To complete the quantization scheme, we leverage a factor τ to scale the input and output and make the quantization function zero-centered by using the offset $\frac{1}{2}\sum_j \alpha_j$, similar to [Yang *et al.*, 2019]:

$$\bar{\mathcal{S}}(w_i) = \frac{1}{\tau}\left(\sum_j \alpha_j \mathcal{S}(\tau w_i - z_j) - \frac{1}{2}\sum_j \alpha_j\right). \quad (5)$$

Recently, [Jin *et al.*, 2019] and [Yu *et al.*, 2019a] adopt the quantization function of DoReFa-Net to achieve a similar goal, and both these methods adopt switchable batch normalization for different bits. We note that these quantization methods are not truly nested, as the quantization function in DoReFa-Net is not nested in terms of different bits. To see this, note that the 2-bit and 3-bit quantization in DoReFa-Net correspond to quantization levels of $\mathcal{Q}_2 = \{-1, -\frac{1}{3}, \frac{1}{3}, 1\}$ and $\mathcal{Q}_3 = \{-1, -\frac{6}{7}, \ldots, -\frac{1}{7}, \frac{1}{7}, \ldots, \frac{6}{7}, 1\}$ respectively. $\mathcal{Q}_3$ does not include all the elements in $\mathcal{Q}_2$. AdaBits [Jin *et al.*, 2019] modify the DoReFa-Net to address this issue.

We compare the quantization function of AdaBits and our proposed FN^3 in Fig. 2, where the coinciding regions are also visualized. In AdaBits, adding more bits is only backtracking the previous lower-bit quantization function, which undermines the previous mapping. If the weight value indicates the importance of its corresponding feature, then once a weight has a maximal quantization value (+1 or -1), then increasing its importance requires changing bits on all other weights. Thus, in AdaBits, the disordered coinciding regions make the optimization harder, thus switchable BN (multiple sets of BN parameters) is adopted, leading to a semi-nested network. In contrast, in FN^3, adding more bits augments information of the previous quantization function, while keeping the steps in the middle unchanged. Adding a new bit can only increase the magnitude of weights, i.e., the corresponding feature's importance. This allows a more natural ranking of the features as the number of bits increases, and better fits our formulation of using additive blocks.

From the regularization perspective, using a limited number of bits for FN^3 quantization also limits the magnitude of the weights, which is consistent with the goal of regularization. In contrast to AdaBits, with low bit numbers, the weights are either maximally activated (+1) or not, which counter-productive for regularization.

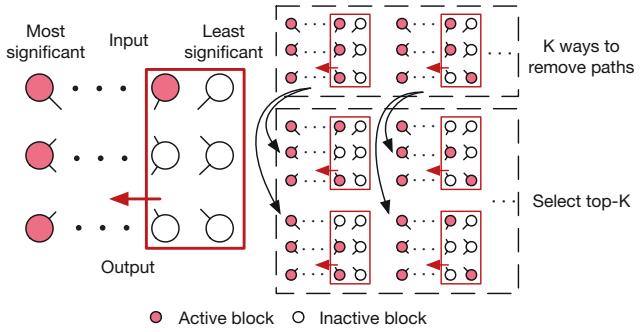

Figure 3: Sketch of the heuristic search algorithm.

3 Heuristic Search

In the hardware-aware NAS papers [Cai *et al.*, 2018; Jiang *et al.*, 2019], given fixed amount of resources, the algorithms search for the best architecture that fits the budget. The algorithm needs to be re-run once the resource budget changes.

Algorithm 1 Heuristic search algorithm

Input: A trained FN^3; candidate number C; block number per layer $\{M_i\}_i$; $\Delta_{\mathcal{P}}$; $\mathcal{P} = 100\% - \Delta_{\mathcal{P}}$; $T = \{\text{tj}_k\}_k^K$; $\text{tj}_k = \emptyset$; pool $= \emptyset$; $\mathcal{W} = \{(i', j')\}$.

while $\mathcal{W} \neq \emptyset$ **do**
  for $k = 1$ **to** K **do**
    for $c = 1$ **to** C **do**
      sample a new candidate (i^*, j^*) from $\mathcal{W}$.
      pool $=$ pool $\cup (\text{tj}_k \cup (i^*, j^*))$.
    end for
  end for
  Evaluate acc. of tj $\in$ pool and select top-K $\{\text{tj}_k^*\}_{k=1}^K$.
  $T = \{\text{tj}_k^*\}_{k=1}^K$.
  $\mathcal{W} = \mathcal{W} \setminus (\mathcal{W} \cup T)$.
  while $\forall j \in \mathcal{W} < (\mathcal{P} + \Delta_{\mathcal{P}})M_i$ **and** $\mathcal{P} \neq 0$ **do**
    $\mathcal{P} = \mathcal{P} - \Delta_{\mathcal{P}}$; Move $\mathcal{W}$ by $\Delta_{\mathcal{P}}$.
  end while
  pool $= \emptyset$.
end while

With a trained FN^3, we can also search for the best architectures for all possible budgets beforehand, at the minimal granularity. We propose a heuristic search method to find the optimal trade-off curve of accuracy w.r.t. number of

blocks, as shown in Alg. 1. The main idea is to maintain multiple records of removed blocks, which we denote as "trajectories". The next block to be removed is selected based on the current trajectories with high accuracy, and thus yields an new trajectory that is more likely to be accurate. Specifically, we maintain a set of K trajectory sets T during searching. $\mathrm{tj}_k = \{(i, j)\} \in T$ is the k-th trajectory set that contains the layer index i and block index j of removed blocks. In every iteration, for one trajectory, C candidate block are sampled from the candidate set and added to the trajectory set. The accuracy of all candidates sets are evaluated and the top-K augmented trajectory sets are selected to update T. As the blocks are highly ordered in FN^3, we leverage a sliding window, $\mathcal{W}$, as the candidate set to restrict the search space, as shown in Fig. 3. The search complexity is reduced from $\sum_i \binom{N}{i} = 2^N - 1$ to KN.

Note that the number of blocks might be different in different layers, e.g., layer 1 has 128 neurons while layer 2 has 512 neurons. We assign the same proportion ($\Delta_{\mathcal{P}}$) of blocks in a sliding window for different layers, e.g., 32 neurons for layer 1 and 128 neurons for layer 2 (25%).

4 Related Work

The idea of nested architecture of neural network was explored by [Baram, 1988], where either square and hexagonal structures are studied. The proposed structures are no longer suitable for modern deep learning tasks.

Adaptive channel: Aiming at adaptive channel and neuron, [Kim *et al.*, 2018] propose a NestedNet to build an n-in-1 network structure by solving weight connection pruning problem iteratively. This work is limited in flexibility as it provides limited number of sub-networks (3 to 4 levels shown in the experiment), while our work has 2^{7136} with MobileNet-V2 in terms of adaptive channel. Our work also outperforms [Kim *et al.*, 2018]; on Cifar10/100, accuracy of [Kim *et al.*, 2018] starts decreasing when half channels are pruned, while our model starts losing accuracy when over 65% components are dropped.

[Yu *et al.*, 2019b] proposed slimmable NN that trains a network with multiple setups of channel number (called width) simultaneously, where the weights are shared among different widths. The network switches to another set of batch normalization (BN) parameters once the width changes thus $\mathcal{O}(M'^2)$ BN parameters should be maintained, where M' is the number of channels in a layer. This method is extended (US-Net [Yu and Huang, 2019]) to allow more choices of width by only recollecting the BN parameters after training. However, it still requires training multiple widths per iteration, which increases the training time. The widths are consistent for different layers, and therefore the number of sub-networks is limited. By comparison, FN^3-channel samples different number of channels for different layer per iteration, increasing the number of sub-network by 2^{N-M} and indicating a higher level of granularity, where $N = \sum_i M_i$. In addition, the increased number of sub-networks in FN^3-channel does not come at a price of increasing training time, making our method more efficient than the slimmable NN.

In FN^3, we observe that, as the importance of blocks keeps decreasing in expectation, the influence of dropping blocks on BN parameters is trivial. Also as observed from the scaling factor during inference $\beta_m = \frac{M+1-m}{M}$ (Sec. 2.2), with m growing larger, the activation is scaled to be smaller, which has low influence on BN statistics. Thus, it is not required to modify the BN scheme in FN^3.

Adaptive bit: [Jin *et al.*, 2019] proposed Adabits that is adaptive in different quantization bits. A quantization scheme similar to DoReFa-net was proposed and the switchable BN method [Yu *et al.*, 2019b] was adopted, thus the number of sub-networks is limited. We also show the limitation of the Adabits quantization function in Sec. 2.4.

More importantly, the order of either channels or bits was not studied in the previous works. We explicitly show that each building block in FN^3 maximizes the incremental information gain (see Sec. 2.3). The analysis validates the idea of using residual additive blocks to build the adaptive neural network. Furthermore, the previous methods only train an adaptive network that provides sub-networks, but it is unclear what is the optimal sub-network given fixed hardware resources. In FN^3, a fast heuristic search method that leverages the block ordering is proposed. A trade-off curve of accuracy w.r.t. the number of blocks can be obtained before deployment, with a low cost. Given a fixed budget of resources, the optimal architecture can be simply obtained by checking the trade-off curve and table lookup[3].

5 Evaluation

The experiments are conducted on the MNIST, Cifar10, Cifar100 and ImageNet datasets. The FN^3-neuron experiment is conducted on MNIST. On Cifar10/100, we run the ablation study of FN^3-layer/-path/-channel/-bit, as well as the the comparisons of FN^3-bit and Adabits. On ImageNet, we run the experiments FN^3-channels to compare with the US-Net. For FN^3-bit, ODO is realized by adding/dropping Heaviside step functions that constitute different quantization functions. The weights in one layer share a quantization function. Thus, in the heuristic search, the block index is replaced with index of Heaviside step function. For the rest 4 block types, ODO is realized by generating masks of ones and zeros onto the input or output of a block. We run the heuristic search on the training data to get the block order. Note that during searching, one number of block might correspond to multiple results (e.g., 50 accuracy results for "100 channels removed"). We calculate the maximum, mean and standard deviation and show them in the graph.

Overview: We show the overview of experimental results and ablation study in Fig. 4. The detailed experimental setups are seen in Sec. A.

The blue curves show the accuracy w.r.t. removed building blocks with normal training. The removed building blocks are randomly selected and each random test is repeated for 100 times. In all the tested cases, the accuracy decreases drastically as building blocks are removed.

The orange curves show the results of ODO training and random removal. On FN^3-layer/-neuron/-bit, the differences

[3]OVIC Benchmark, Tensorflow.

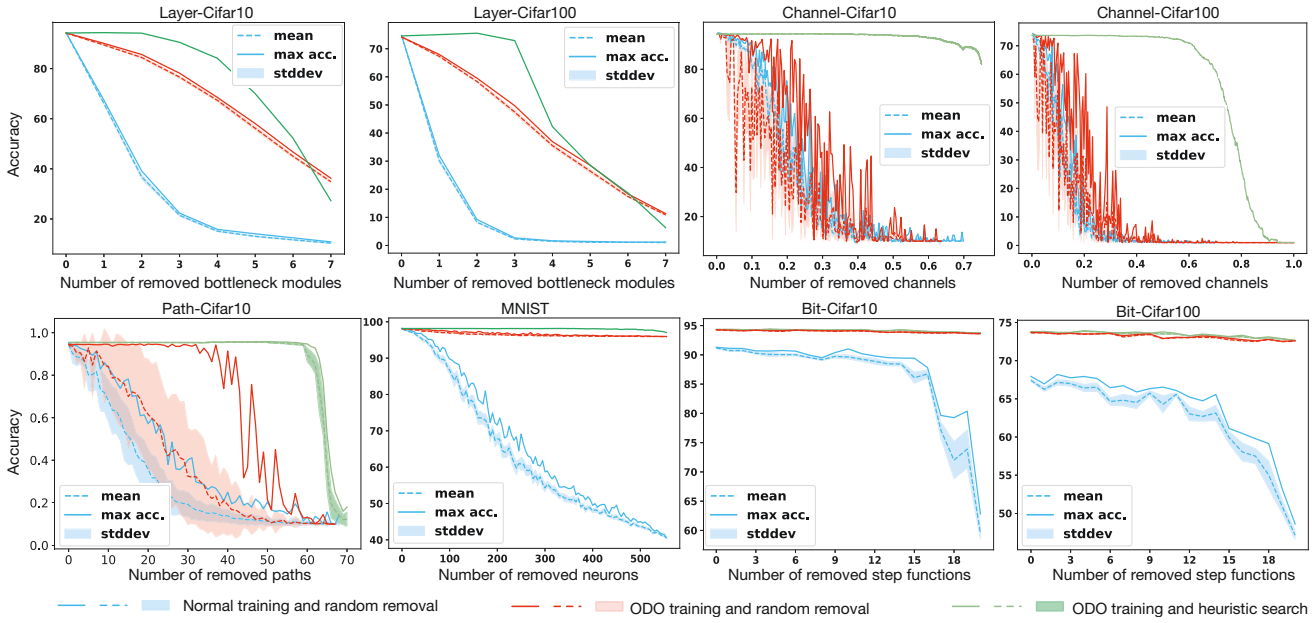

Figure 4: Experimental results for FN^3-layer, FN^3-path, FN^3-channel and FN^3-bit on Cifar10/100, FN^3-neuron on MNIST.

Model	#sub-networks	Training epochs
US-Net	2^{26}	$250 \times n$
FN^3-channel	2^{7136}	400

Table 1: Numerical comparisons.

between orange curves and blue curves are significant. This indicates ODO effectively improves the overall classification performance. On FN^3-bit, ODO also improves the quantization performance of the *whole* network, as observed from the difference of starting points of the orange and blue curves. As for FN^3-path, the mean of accuracy has a similar trend as normal training, while the max accuracy starts decreasing when around 40 paths are removed. Therefore, ODO effectively generates highly accurate and small sub-networks. The orange curve of FN^3-channel shows a similar behavior as FN^3-path's while the fluctuation of max accuracy is more severe, which indicates the unstable performance of random removal.

The green curves shows the results of ODO training and heuristic search. In general, the heuristic search effectively finds a better trade-off curve. Moreover, as observed from the results of FN^3-channel and FN^3-path, the variance is significantly reduced and useless trajectories are pruned, thus the search complexity is reduced. This indicates the heuristic search can discover good sub-networks efficiently. The heuristic search also helps to reduce the number of evaluations during searching. For example, on FN^3-path, the number of evaluations in random removal is 7200, while that in heuristic search is 1672, reduced by 76.8%.

Next, we discuss FN^3-channel and FN^3-bit in detail and show the comparisons with previous works.

FN^3-channel: We train FN^3-channel with MobilenetV2 on Imagenet. The ODO mask is multiplied with the output of

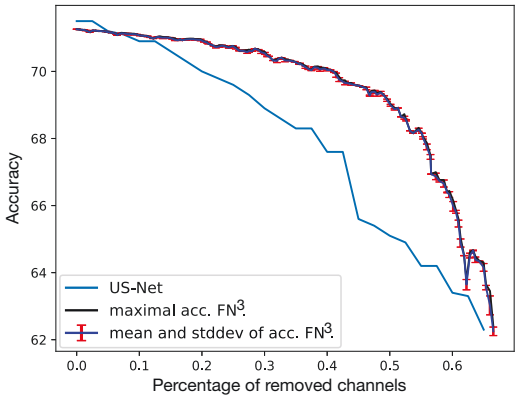

Figure 5: Performance of FN^3 with MobileNetv2 on ImageNet.

depth-wise convolution layer in each inverted residual block. Therefore, the zeros in the mask disable the corresponding filters of preceding point-wise convolution, depth-wise convolution and the channels of subsequent point-wise convolution [Sandler *et al.*, 2018].

Fig. 5 shows that FN^3 achieves a better trade-off of accuracy w.r.t. removed blocks compared with US-Net. This reveals the superiority of ODO over other methods that do not consider order of blocks. As shown in Table 1, as FN^3 is fully nested, the theoretical number of sub-networks is much greater than that of US-Net. The training epochs of FN^3 is steady at 400, while that of US-Net is $250n$, where n is the number of sampled widths per iteration (normally $n \geq 4$).

FN^3-bit: The experiments on quantization are run on Cifar10/100 datasets with MobileNet-v2. The model is optimized with Adam and a cosine annealing scheduler for 600 epochs. In FN^3-bit, we use a quantization level schedule similar to [Yang *et al.*, 2019], 2-bit: $\{-1, 0, 1\}$, 3-bit($\pm$2): $\{-2, -1, 0, 1, 2\}$, 3-bit($\pm$4): $\{-4, -2, -1, 0, 1, 2, 4\}$, 4-bit:

Cifar10	4-bit	3-bit (± 4)	3-bit (± 2)	2-bit
Adabits	91.84	79.3	-	58.98
FN^3-bit	94.28	94.2	94.2	93.64
Cifar100	4-bit	3-bit (± 4)	3-bit (± 2)	2-bit
Adabits	72.15	65.23	-	51.67
FN^3-bit	76.65	76.34	75.35	74.97

Table 2: Accuracy for the same bit-width for whole network.

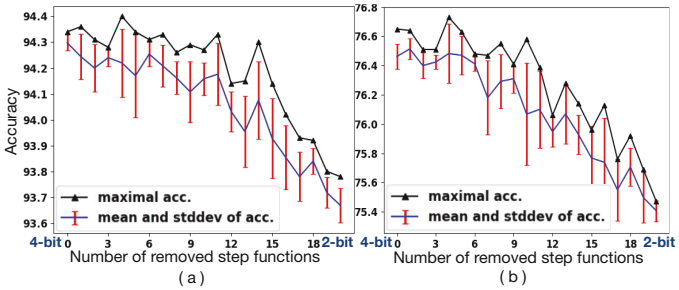

Figure 6: Performance of FN^3-bit using MobileNetv2 with removed step functions (bits). (a) Cifar10. (b) Cifar100.

$\{-8, -4, -2, -1, 0, 1, 2, 4, 8\}$. τ is initialized to $\frac{5p}{4q}$, where p is the max absolute quantization interval and q is the max absolute value of weight. To reproduce the Adabits method, we use the modified DoReFa-net quantization function and switchable BN parameters. Following [Jin *et al.*, 2019], during training of Adabits, we progressively reduce the number of bits, with each bit-width taking 150 epochs.

The results of using the same bits for whole network are shown in Table 2. The performance of Adabits on lower bits drop dramatically, while the performance of FN^3-bit only reduces by 0.64% from 4-bit to 2-bit. This validates the idea of composing a nested quantization with multiple step functions.

In FN^3, we apply different bit-widths to 7 groups of layers which leads to $4^7 = 16384$ sub-networks while Adabits only provides 3 sub-networks. Thus, with FN^3, different layers can use different bit-widths for more flexible deployment to the given resource constraints, which is not possible with Adabits. Fig. 6 shows the results using mixed bit-widths between layers, obtained by running the heuristic search on the model used in above experiment. The x-axis indicates the bit usage, where the left side corresponds to every layer using 4 bits, the right side is every layer using 2 bits, and in between are mixed bit-widths with different number of bits (or step functions) removed. The accuracy w.r.t. bit-width curve shows that a mixed bit-width can also achieve similar accuracy to using 4-bits for all layers. For example, using around 3 bits (9 step functions removed) has similar performance to using full 4-bits.

6 Conclusion

We propose the fully nested neural network that generates large amounts of nested sub-network, based on additive network building blocks. The proposed heuristic search is able to efficiently find the optimal sub-network, and experiment results show that our method is widely applicable to compress/quantize a variety of NN building blocks. Future work will consider how to combine different levels of blocks (e.g., channel and bits) to generate more adaptive networks.

A Detailed Setup of Experiments

FN^3-layer: We run standard RexNeXt-Cifar [Xie *et al.*, 2017] on Cifar10 and Cifar100 dataset. ResNeXt-Cifar contains one convolutional layer for feature extraction which is kept active in this experiment, and 9 bottleneck modules each being treated as a block. We assign a uniform probability $\frac{1}{9}$ to the Categorical distribution $\mathcal{C}(m)$ where each dimension corresponds to the index of bottleneck module. Therefore, the first bottleneck module is the least likely to be dropped. We use SGD with an initial learning rate 0.1, momentum factor 0.9 and batch size 128. The learning rate is scaled by 0.1 at epoch 150 and 225. The network is trained for 300 epochs. For heuristic search, as the ordered dropout (ODO) is performed on the bottleneck modules (group of layers), the height of sliding window is one. Because these blocks are well ordered, we set the window width to be 1 ($\Delta_{\mathcal{P}} = \frac{1}{9}$). In other words, the algorithm greedily selects the next block to be removed.

FN^3-path: We run the same RexNeXt-Cifar as in FN^3-layer on Cifar10 and Cifar100 dataset. In each bottleneck module, 8 paths are treated as the blocks. We assign a uniform probability 1/8 to the Categorical distribution $\mathcal{C}(m)$ where each dimension corresponds to the index of blocks. 9 ODO units operate simultaneously. The rest hyper-parameters are the same as FN^3-layer. In such case, for heuristic search, the height of sliding window is 9. $\Delta_{\mathcal{P}} = 12.5\%$, C=10 and K=3.

FN^3-channel: To compare with previous works, this experiment runs with MobileNet-V2 on ImageNet dataset. The ODO mask is applied to the depth-wise convolution layer in each inverted residual block. Therefore, it disables the filters of former point-wise convolution, depth-wise convolution and the channels of subsequent point-wise convolution. We keep 35% channels active during this experiment, for a fair comparison. As there are 17 inverted residual blocks, 17 ODO units operate simultaneously. The model is optimized with Adam optimizer with initial learning rate 0.1 and a cosine annealing scheduler for 400 epochs, on 6 GTX 2080ti GPUs. In each forward pass of the network, for each ODO unit, the sampled index $m \sim \mathcal{C}(\cdot)$ is broadcasted to all GPUs for consistency. For heuristic search, the height of sliding window is 17. $\Delta_{\mathcal{P}} = 12.5\%$, C=10 and K=3. On Cifar10/100, similar settings are adopted.

FN^3-neuron: We run an MLP of structure 784-512-128-10 on MNIST. We keep 12.5% neurons active during this experiment. Two ODO units are applied on the rest neurons in two hidden layers, with uniform probabilities $\frac{1}{448}$ and $\frac{1}{112}$ respectively. The network is trained with SGD with an initial learning rate 1.0 for 100 epochs. The learning rate is scaled by 0.7 every 10 epochs. For heuristic search, we set the height and width of window to be 1, similar to FN^3-layer.

The setups for FN^3-bit are shown in Sec. 5.

References

[Baram, 1988] Yoram Baram. Nested neural networks. *NASA Technical Memorandum 101032*, 1988.

[Cai *et al.*, 2018] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[Han *et al.*, 2016] Song Han, Huizi Mao, and William Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

[Hastie *et al.*, 2009] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[Jiang *et al.*, 2019] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 5. ACM, 2019.

[Jin *et al.*, 2019] Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. *arXiv preprint arXiv:1912.09666*, 2019.

[Kim *et al.*, 2018] Eunwoo Kim, Chanho Ahn, and Songhwai Oh. Nestednet: Learning nested sparse structures in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8669–8678, 2018.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[Neklyudov *et al.*, 2017] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[Rippel *et al.*, 2014] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754, 2014.

[Sandler *et al.*, 2018] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[Veit *et al.*, 2016] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pages 550–558, 2016.

[Wang *et al.*, 2019] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[Xie *et al.*, 2017] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[Yang *et al.*, 2019] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7308–7316, 2019.

[Yu and Huang, 2019] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019.

[Yu *et al.*, 2019a] Haichao Yu, Haoxiang Li, Honghui Shi, Thomas S Huang, and Gang Hua. Any-precision deep neural networks. *arXiv preprint arXiv:1911.07346*, 2019.

[Yu *et al.*, 2019b] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019.

[Zhang *et al.*, 2018] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[Zhou *et al.*, 2016] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

[Zhou, 2012] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.