URNet : User-Resizable Residual Networks with Conditional Gating Module

Sangho Lee,[*1] **Simyung Chang,**[*12] **Nojun Kwak**[†1]
[1]Seoul National University, Seoul, Korea
[2]Samsung Electronics, Suwon, Korea
{shlee223, timelighter, nojunk}@snu.ac.kr

Abstract

Convolutional Neural Networks are widely used to process spatial scenes, but their computational cost is fixed and depends on the structure of the network used. There are methods to reduce the cost by compressing networks or varying its computational path dynamically according to the input image. However, since a user can not control the size of the learned model, it is difficult to respond dynamically if the amount of service requests suddenly increases. We propose User-Resizable Residual Networks (URNet), which allows users to adjust the computational cost of the network as needed during evaluation. URNet includes Conditional Gating Module (CGM) that determines the use of each residual block according to the input image and the desired cost. CGM is trained in a supervised manner using the newly proposed scale(cost) loss and its corresponding training methods. URNet can control the amount of computation and its inference path according to user's demand without degrading the accuracy significantly. In the experiments on ImageNet, URNet based on ResNet-101 maintains the accuracy of the baseline even when resizing it to approximately 80% of the original network, and demonstrates only about 1% accuracy degradation when using about 65% of the computation.

Introduction

Generally, the computational graph in a deep neural network is fixed and unchanged during inference time. But in many situations of real applications, there may be the case that the system needs to handle various amounts of computation per request (Herbst, Kounev, and Reussner 2013). For example, in the situation that the number of requests is rapidly increasing but the system is forced to respond quickly, it is better for the system to dynamically allocate less resource for requests within a moderate performance degradation bound.

There are many researches that suggest static compressed model(Hinton, Vinyals, and Dean 2015; Howard et al. 2017; Iandola et al. 2016). Unlike these works, recent researches (Wu et al. 2018; Lin et al. 2017) suggest the methods that a neural network dynamically changes its computation graph at test time, rather than fixed all the time. But these works only change the network path for each input, e.g., easy samples follow the path with less computation but complex samples require maximum available computation. Therefore, these works can not take care of the demand from the external environment. They are dynamic but cannot resize on our own purpose.

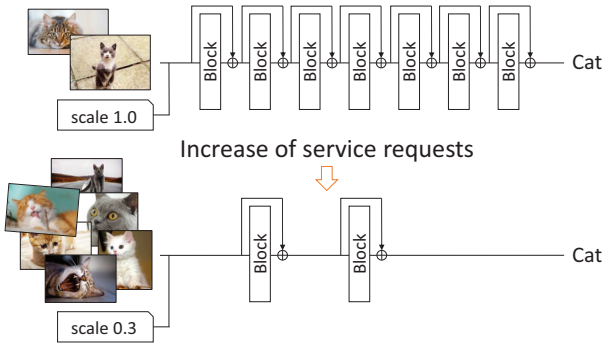


Figure 1: The concept of URNet. Our method uses the entire network when resources are sufficient. If the number of service requests increases, the system or a user can change the **scale(the size of the computational cost)** of the network to use only a fraction of the entire blocks, thereby reducing the amount of computation in the network and processing the increased requests in time.

In this paper, we suggest a model that can adjust its computational cost by dropping some of its components. It follows given user's demand by itself, like 70% or 50% of maximum resource for usage, at any inference time. Our model is also variant to input samples, but its computational cost does not deviate significantly from the desired one. It is robust to the environment where the resources per request are limited or dynamically changing over time, and therefore, it fits such applications as in a backend server or background applications in a client. Figure 1 intuitively describes our concept. Our model is basically a plain ResNet (He et al. 2016) architecture with additional gate modules located between neighboring blocks. Our gate module is computationally very cheap compared to the backbone network. Like the works

[*]These authors also contributed equally to this work
[†]This author is a corresponding author

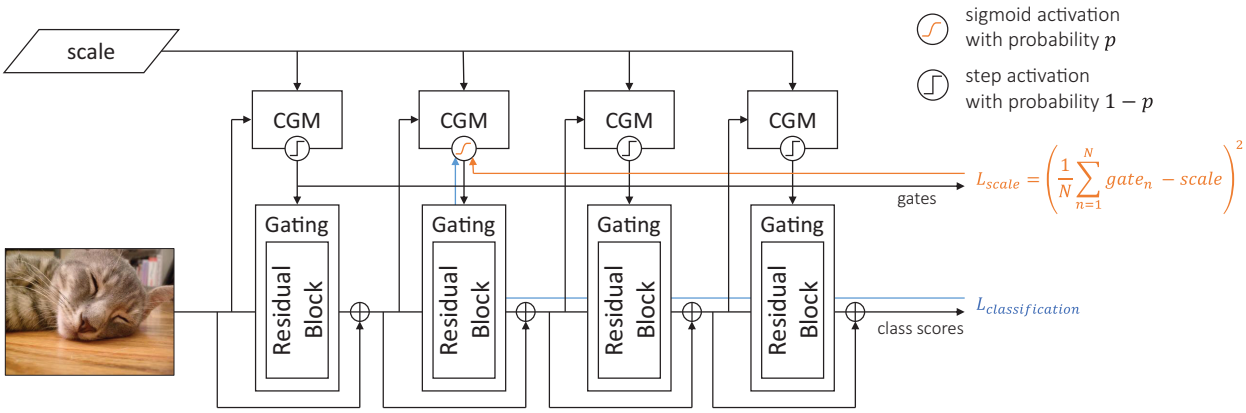


Figure 2: Overall structure of URNet. URNet locates Conditional Gating Module (CGM) for each residual block of ResNet, and determines whether each block is used or not. CGMs receive two inputs and output one gate value: one of the inputs is the features of the previous layer and the other is the desired scale parameter $\mathcal{S}$. To output a gate value, it uses a sigmoid function with probability p and a binary step function with probability $1-p$ as an activation function. CGMs, with sigmoid function, can be trained through *scale loss* L_s and classification loss L_c to increase classification performance while controlling the number of blocks used. At the time of inference, p is set to 0, so that only the binary step is used and the amount of computation is reduced by not using the blocks with the gate value of 0.

in (Mirza and Osindero 2014; Sohn, Lee, and Yan 2015; Chen et al. 2016), these modules are conditional, that is, the user-specified scale condition of a network can be fed into them. The network actually adjusts its scale by dropping some blocks of ResNet according to the binary output of the conditional gating module.

To train the network so that it can gate the corresponding block, we may necessarily need to use 0/1 binary valued function which decides whether to use the component or not. Since this binary valued function is not differentiable, this problem has usually been approached with reinforcement learning (Wu et al. 2018; Lin et al. 2017). On the other hand, we use a sigmoid function instead as a differentiable substitute of a binary function, and train the whole model while taking each gate module as either a sigmoid or a binary function based on a probabilistic rate. In addition, since our training losses can be implemented from the conventional classification loss by just adding mean squared error loss between the desired input scale parameter $\mathcal{S}$ and the actual network scale, we do not need reinforcement learning, and the training is fast and stable.

We validate our model from the experiments on CIFAR-10, CIFAR-100 (Krizhevsky 2009) and ImageNet (Deng et al. 2009) datasets. By experiments, we show that our method can fit its scale to a given condition well, and sometimes outperforms the baseline ResNet model when the scale parameter $\mathcal{S}$ is 70% or 80%. Furthermore, even if we only use 60% of blocks, the accuracy does not severely degrade.

Our contributions are summarized as follows:
(1) We propose URNet that can control its computational complexity and inference path according to the user's demand.
(2) URNet does not suffer much from performance degradation even if it reduces the amount of computation.
(3) URNet is able to learn non-differentiable binary gates using a supervised learning method instead of using reinforcement learning, thus improving learning speed and stability.

Related Works

Model Compression There are many works on compression of neural networks, like pruning(Luo, Wu, and Lin 2017; He, Zhang, and Sun 2017; He and Han 2018; Yu et al. 2017), architecture search(Pham et al. 2018; Zoph and Le 2016), model designing(Howard et al. 2017; Iandola et al. 2016; Chen et al. 2018). Most of these compression methods produce one static-sized model. To cope with the requirements of dynamically changing environment, it should prepare many different-sized networks to its memory, which is not desirable for a resource-constrained scenario like in an embedded environment.

Rule-based Dynamic Network To cope with those dynamic demand, there are several methods who can cut away the computational graph, like early stopping(Huang and Chen 2018), or channel throwing(Yu et al. 2018). In inference time those methods can choose fixed subset of full computational graph that gradually shallower with depth, or narrower with channel, on predefined rule-based points of computation amount, like $\{0.5, 0.75, 1.0\}$.

Learning-based Dynamic Network Unlike those rule based methods that value of computation amount is not continuous and selected sub-graph is fixed under architecture, there are another approaches that selection of subgraph is decided by model itself(Lin et al. 2017; Wu et al. 2018; Liu and Deng 2018; Odena, Lawson, and Olah 2017; Bolukbasi et al. 2017). Those methods have variance of computational graph for inputs, so more dynamically flexible than rule based one. But to make the model to decide itself, these work usually encounter the problem of handling the non differentiable function(e.g. binary gate) on deciding hard gates. In many times this approaches are achieved by reinforce-

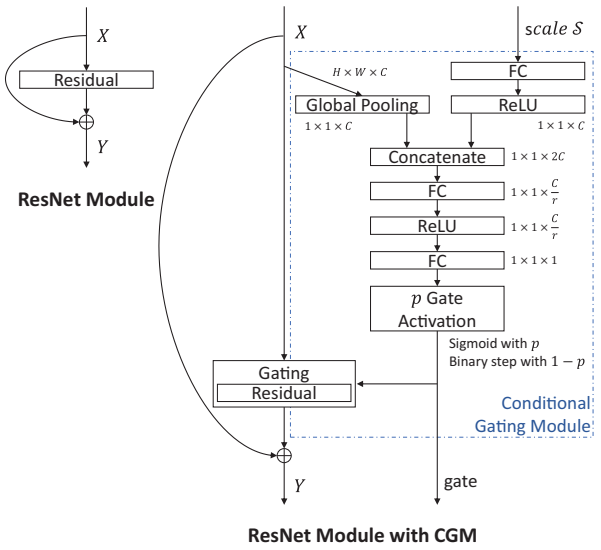


Figure 3: The structure of Conditional Gating Module (CGM). The left is the ResNet module, and the right side is the ResNet module with CGM. CGM is a module that receives an input feature of residual block and a scale parameter $\mathcal{S}$, then outputs a gate in a sigmoid or binary form through a lightweight mapping function.

ment learning, which normally encounters the problem of slowness and instability.

Our method can achieve both two dynamic approach's main goal, learning-based dynamic network with user constraint, with the benefit of variance per samples under any continuous valued resource constraint, without using reinforcement learning.

User-Resizable Residual Networks

Our goal is to train a network to adjust its size according to the given *desired scale parameter* $\mathcal{S}$ with a constraint of spontaneously minimizing the performance degradation. $\mathcal{S}$ can be any value between 0 and 1, representing what amount the user wants to scale the network. To achieve this, we propose *User-Resizable Residual Networks* (URNet) and a training method for them. Figure 2 is the overview of our URNet. We use ResNet as a baseline network, and drop the residual blocks spontaneously upon $\mathcal{S}$ to scale the network. For this, URNet includes a *Conditional Gating Module* (CGM) as a method to decide whether to use each block or not. This module is located one for each residual block, and outputs a gate value under the condition of input feature and the scale parameter $\mathcal{S}$. To train the gates with the conventional supervised learning method, we propose a *scale loss* and its training method. More specifically, because binary gates can not be back-propagated due to its non-differentiable characteristics, our gate has either a sigmoid or binary form at a certain probability during training. However, at the time of inference, it is always set as a binary step function.

Conditional Gating Module

To determine whether to use each block of a ResNet, some modules or separate networks are required. And it must be decided before each block is activated. We propose a conditional gating module (CGM), a lightweight network module for this purpose. CGM is a simple structure that can be embedded in a ResNet, which can effectively determine whether to use a block or not with much fewer parameters and computation compared to those of the main ResNet.

As shown in Figure 3, this gate module has two input entries, one is related to the features in the previous layer ($X \in \mathbb{R}^{H \times W \times C}$), and the other is the scale parameter $\mathcal{S}$. Thus, the gate module is conditional to both X and $\mathcal{S}$. Several studies (Lin, Chen, and Yan 2013; Hu, Shen, and Sun 2018; Chang et al. 2018) have used global pooling to handle global features with fewer operations, and the proposed CGM also uses a global average pooling to handle the features of the previous layer. CGM then concatenates the globally pooled features of size $1 \times 1 \times C$ with $\mathcal{S}$ expanded to the same size, and outputs one gate variable through several fully connected layers and activation functions. In order to reduce the amount of computation, we use a reduction rate r after the concatenation of the global feature and the scale condition like SENet (Hu, Shen, and Sun 2018).

For scalability at inference time, we use *Gate-Activation* function followed by several fully connected layers. *Gate-Activation* is a simple function we designed for CGM, which works as either a sigmoid function or a binary step, depending on the given probability. CGM can learn only when the gate is sigmoid where gradients can be calculated and back-propagated. We call this frequency as a gate-training probability p. The Gate-Activation operates as a sigmoid function with probability p and acts as a binary step function with probability $1 - p$. In this case, training with binary gates plays a very important role as well. When all gates are activated with a sigmoid function, the remaining blocks cannot learn properly the cases of not using specific blocks. During the evaluation, this p is fixed to 0 and only the binary function is used as an activation.

This gate module is then incorporated within the ResNet. Generally, the ResNet Module can be defined as:

$$Y = X + F(X), \tag{1}$$

where X and Y are the input and the output vectors of each layer, and the function $F(X)$ represents the residual mapping to be learned. The operation $X + F$ represents an element-wise addition as a shortcut connection. Since the purpose of CGM is to gate the output of the function $F(X)$, the ResNet module with CGM can be expressed as:

$$\begin{aligned} Y &= X + F_{gating}(F(X)) \\ &= X + CGM(X, \mathcal{S}) \cdot F(X) \\ &= X + gate \cdot F(X). \end{aligned} \tag{2}$$

F_{gating} refers to block-wise multiplication between $F(X)$ and a $gate$ which is the output of $CGM(X, \mathcal{S})$. When the Gate-Activation works as a binary gate during evaluation, the module can be expressed as:

$$Y = \begin{cases} X, & \text{if } gate = 0 \\ X + F(X), & \text{otherwise.} \end{cases} \tag{3}$$

As shown in (3), the corresponding block can be dropped if the value of the gate is 0. The computation of the block can then be reduced because $F(X)$ operation is omitted. For small $\mathcal{S}$, the gate values will have a good chance to be 0, depending on the input feature map X, resulting in a reduced computational complexity on average. On the other hand, large $\mathcal{S}$ will mostly activate gates such that most blocks will be used for inference, resulting in high performance.

Training CGM

Scale Loss It was mentioned that CGM can output a sigmoid or binary gate, and can be learned through back propagation when using sigmoid gates. However, for actual learning, an objective function must be defined. The goal of our method is not only to increase or maintain the performance of the classification, but also to allow the user to change the size of the network according to the desired one. Thus, the objective function must also satisfy both of these requirements. We propose a *scale loss* that can be used with conventional supervised learning methods. This loss is defined so that the average of CGM gates is close to the scale parameter $\mathcal{S}$, as follows.

$$L_s = ((\frac{1}{N}\sum_{n=1}^{N} gate_n) - \mathcal{S})^2, \tag{4}$$

where N denotes the number of residual blocks in the URNet and $gate_n$ represents the output of the CGM corresponding to the n-th block. The full objective of URNet is the sum of this *scale loss*, L_s, and the classification loss (cross entropy), L_c, of ResNet:

$$L = L_c + \beta L_s. \tag{5}$$

Here, β is a hyper parameter that controls the weights of L_c and L_s. Smaller β means a bigger weight on classification, while bigger β means a bigger weight on the scale loss. As β increases, the actual block usage becomes similar to $\mathcal{S}$, but the classification performance may be sacrificed somewhat. Our experiments show that the number of actual blocks used can be controlled to be very close to the scale parameter $\mathcal{S}$.

Gate Training Scheme According to (5), the CGMs are optimized to increase the classification performance and to make the average value of gates similar to the input parameter $\mathcal{S}$. However, in order to ensure that URNet operates at various values of $\mathcal{S}$ during inference, these values must be learned during training. This is done by randomly changing the range of $\mathcal{S}$ as we want to resize. The distribution of $\mathcal{S}$ is set as a uniform distribution of $\mathcal{U}(\mathcal{S}_{min}, \mathcal{S}_{max})$ during training. Here, $\mathcal{S}_{min}$ and $\mathcal{S}_{max}$ are the minimum and maximum of the range, respectively. Through this, the value of $\mathcal{S}$ and the actual block usage are synchronized with each other.

Since we use a pre-trained ResNet as the base network, we train only the CGM first, similar to BlockDrop (Wu et al. 2018) which trains the policy network first. This is to minimize the influence of premature CGM on the pre-trained ResNet. After then, ResNet and CGM are jointly trained. However, by using the supervised learning method, CGM can be learned directly without using the method like the curriculum learning (Bengio 2013) which is used to overcome the instability of reinforcement learning in the BlockDrop paper, and learning can be performed with much less epochs. For CIFAR datasets, our method requires only 500 epochs which is a considerably smaller number compared to the training of BlockDrop which takes a total of 7,000 epochs including curriculum learning of 5,000 epochs.

Experiments

Baselines and Experimental Setup

In the following experiments, we have trained and evaluated our method on CIFAR-10, CIFAR-100 (Krizhevsky 2009) and ImageNet (Deng et al. 2009) datasets with top-1 accuracy. As a base network for our URNet, we have used ResNet-110 (54 blocks) for CIFAR datasets, and ResNet-101 (33 blocks) for ImageNet. We have chosen the channel reduction rate r of CGM (see Figure 3) as 2 for CIFAR datasets and 16 for ImageNet. Similar to the evaluation of other compression methods, we calculate the number of multiply-accumulate operations of convolutional layers and linear layers in FLOPs (floating point operations). The total number of FLOPs of all the CGMs in ResNet-110 is only 0.04% of the base network and 0.08% for the ResNet-101. We train CGM only for 100 epochs on CIFAR datasets and 5 epochs on ImageNet. Then, we train CGM and the base network jointly for 400 additional epochs on CIFAR and 15 epochs on ImageNet. The learning rate is adjusted from 10^{-3} to 10^{-5}.

Result on CIFAR

Table 1 shows the result of our method on CIFAR-10 and CIFAR-100, under various values of scale parameter $\mathcal{S}$. As shown in the table, our method can be resized as desired according to the given value of $\mathcal{S}$, without severe accuracy degradation. The table contains two baseline results of plain ResNet-110 which contains 54 residual blocks. It also contains the results of the proposed URNet (Ours), and other different settings with ablation. For those experiments we have set β in equation (5) as 2.0. During training, the scale parameter $\mathcal{S}$ has been uniformly sampled in the range of [0.2, 1.0], for every iteration.

The first and the second rows show the result of two baseline experiments with ResNet-110. The first row (ResNet-110 with rand, val) is the plain pretrained ResNet but we randomly drop the residual blocks at test time, to resize the network according to the given $\mathcal{S}$. The second row (ResNet-110 with rand, train/val) is the results of the finetuned ResNet that was trained with randomly dropping the blocks. It is not surprising that the performance of the second row is increased compared to the baseline at $\mathcal{S} = 1.0$, because this can be interpreted as the dropout effect applied to block units. This result is very similar to the work in (Huang et al. 2016), as they trained the ResNet with dropping each layer by a specific probability and unified them at test time. What is different from (Huang et al. 2016) is that they trained different drop probability for each blocks but ours is the same for all blocks.

Table 1: The accuracy (%) and the number of block used under various scale conditions $\mathcal{S}$. The two row numbers in each cell are the accuracy (first row) and the number of blocks used (second row). Our method URNet(Ours) can be resized to match the user condition well, without severe accuracy degradation. Compared to the baseline with $\mathcal{S} = 1.0$ (93.2% (CIFAR-10), 72.3% (CIFAR-100)), our method performs better for a wide range of $\mathcal{S}$ ($0.6 \sim 1.0$). The variance of block usage is in the supplementary.

	CIFAR-10					CIFAR-100				
scale parameter $\mathcal{S}$	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
ResNet-110	11.8	15.4	28.0	68.6	93.2	1.3	2.4	7.3	38.1	72.3
(rand, val)	10.80	21.65	32.34	43.21	54.00	10.77	21.56	32.46	43.19	54.00
ResNet-110	83.3	91.0	92.8	93.3	93.7	50.0	66.2	70.8	72.2	73.0
(rand, train/val)	10.80	21.65	32.38	43.16	54.00	10.83	21.58	32.40	43.24	54.00
External network	91.5	92.6	92.7	93.1	93.0	70.3	71.1	71.4	72.5	72.5
(ResNet-8)	31.15	32.30	32.92	47.34	51.00	18.73	21.00	28.56	45.25	53.94
URNet SG	12.7	21.4	74.9	81.7	81.3	2.2	5.3	16.6	20.0	17.7
($p = 1.0$)	6.75	14.05	46.69	53.64	53.88	8.58	12.26	40.58	51.87	53.49
URNet BG	93.2	93.1	93.0	92.9	92.8	71.5	71.6	71.7	71.8	71.7
($p = 0.0$)	28.55	28.60	28.67	28.81	28.84	27.97	28.20	28.54	28.81	29.03
ResNet+B/A	83.1	91.1	92.5	93.3	93.7	50.1	66.4	70.4	72.2	73.2
(rand, train/val)	10.78	21.54	32.42	43.15	54.00	10.79	21.56	32.44	43.22	54.00
URNet(Ours)	92.2	93.3	**93.7**	**93.7**	93.6	70.7	71.5	72.4	**73.0**	72.8
($p = 0.1$)	18.08	20.86	32.02	44.37	52.19	28.10	28.57	32.00	44.61	49.41

The URNet (Ours) is trained with the gate training probability $p = 0.1$, from the pretrained ResNet-110. Our method can match the network size to the desired value of $\mathcal{S}$ very well, without severe accuracy degradation. Note that at sizes in the range between 60% and 100% (32 blocks to 54 blocks), our method can even perform better than the baseline ResNet (93.2% (CIFAR-10) and 72.3% (CIFAR-100)). Unlike the finetuned dropped ResNet in the second row, our method does not severely degrade under very sparse block usage. Our method does not drop the blocks randomly like the compared method in the second row, but it drops the blocks by the decision of CGMs. This can be the reason for the lowered damage, as the CGMs can separate the blocks into most usable blocks and the remainder. As shown in Figure 4, under low $\mathcal{S}$ the CGMs have a tendency to open most important blocks exclusively, and these blocks are opened at every scale. And as $\mathcal{S}$ gets bigger, the rest of blocks gradually start to open (color changes from blue to yellow) because more blocks are getting more affordable.

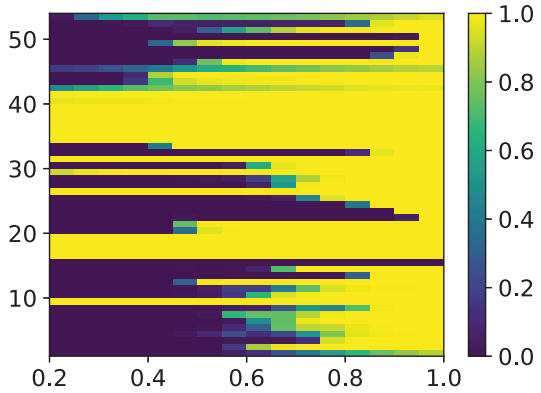
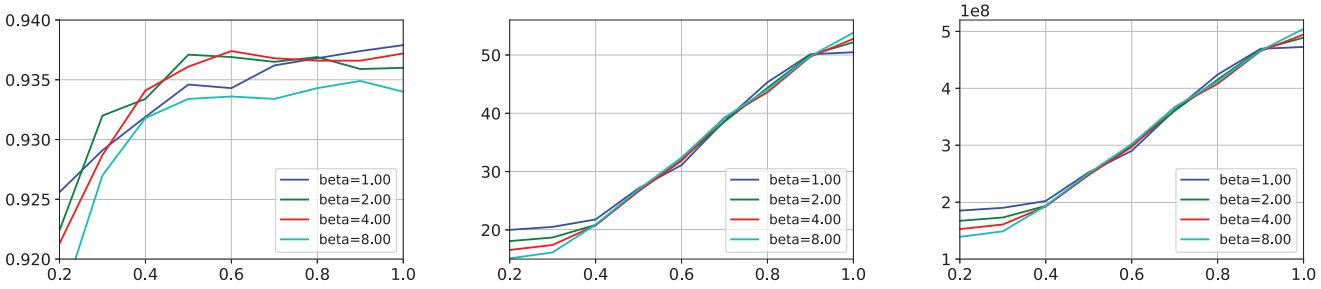


Figure 4: The block usage map of URNet-110 on the CIFAR-10 test set with $\beta = 4.0$. The horizontal axis is the scale parameter $\mathcal{S}$ and the vertical axis is the index of 54 residual blocks. As $\mathcal{S}$ increases, the usage of blocks gradually increases. Also, the presence of blocks whose usage is not 1 or 0, means that the usage of the block varies according to the input image even on the same scale.

Ablation Study In Table 1, there are 4 other experiments for ablation study. The External network method uses an external small network with 3 residual blocks (equivalent to ResNet-8), which is separated from the base network, similar to the method presented in (Wu et al. 2018), but it is not trained using reinforcement learning. This external network is trained similar to the CGMs, by switching between sigmoid and binary activation with a rate of p. However, an important difference is that this module handles all of the gating at once with input data. It needs more computation compared to ours, but it is hard to expect them to extract rich features as it is smaller than the base network. As shown in the Table, the external network method does not work well to meet our purpose and the network usage deviates much from the scale parameter $\mathcal{S}$.

The URNet with sigmoid only (URNet-SG) is a special case of URNet with $p = 1.0$, where the network is never trained with binary activation. But at inference time, all the CGMs are binary activated because our purpose is to drop some blocks. This experiment is a counter example that shows why the binary activation is needed during gate training. It shows that if we gate the block by just using a sigmoid value, the performance degrades severely. The URNet-BG is trained with $p = 0$, which indicates that the network is trained with only binary activations. In this case, the CGMs



(a) Accuracy (b) Block usage (c) FLOPs

Figure 5: Accuracy, Block usage, and FLOPs versus scale parameter $\mathcal{S}$ under various βs, result of URNet-110 on CIFAR-10 dataset. The block usage and FLOPs follow the scale parameter $\mathcal{S}$ well, and better if β is bigger. For accuracy, too big β can downgrade the accuracy, so a moderate value of β can perform better.

are actually not trained and the block features are just multiplied by the untrained CGM output. The URNet-BG experiment shows that without sigmoid activation, the URNet can not resize to the desired size $\mathcal{S}$ at all, and the result is just from an additional training (400 epochs) of an arbitrary subgraph of ResNet.

The ResNet+B/A always uses plain sigmoid function for the activation of CGM, which can be considered that $p = 1.0$ at both train and test time. Note that the variants of URNet set $p = 0.0$ at test time. In this case, it can learn the continuous block-wise attention (0∼1), so possibly it gains more accuracy than the baseline ResNet. However, the ResNet+B/A has no binary function, thus it should calculate all the blocks, which means that it is not resizable. Resizing it with a random drop during training (ResNet+B/A(rand, train/val)) results in similar performance with the second row of the Table. It shows that the model can get accuracy gain with block attention, but suffers such a degradation when trying to resize by applying a random drop. If we force the B/A module output to hard attention by thresholding the continuous attention at test time, it is identical to URNet-SG, which also fails to our purpose. Even if the CGMs in URNet does not utilize the gain from continuous block-attention (0∼1), it outperforms the ResNet+B/A for most values of $\mathcal{S}$. How the URNet does not suffer such degradation (and even gain accuracy) is that it can learn whether the block is necessary or can be abandoned, under given $\mathcal{S}$, by the proposed gate training scheme. As can be inferred from Figure 4.

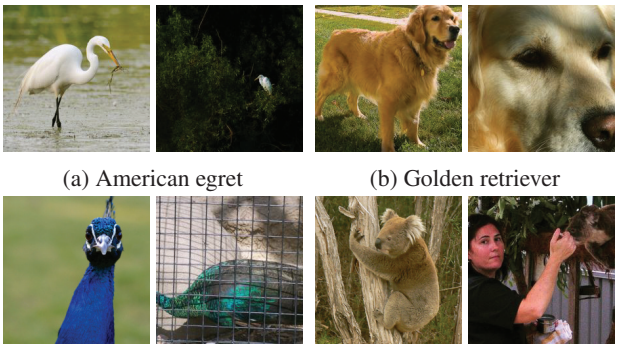
Resize Ability The hyper-parameter β in (5) can represent how strictly we want the network to follow the desired scale $\mathcal{S}$. If we set β higher, the network is more strongly affected by the scale loss. As shown in Figure 5(b), the higher β becomes, the more strict the network becomes in following the target scale. For lower β, the block usage is slowly fixed at the boundary of $\mathcal{S}$, especially when $\mathcal{S} = 0.2$. If β is too big, the accuracy of the network seems to be downgraded as shown in the case of $\beta = 8.0$ in Figure 5 (a). This is because too much scale loss can constrain the network capacity leading to a poor classification loss. But Figure 5 (a) shows that β and the accuracy does not have a complete negative correlation for relatively small β ($\beta = 1, 2, 4$), and the maximum accuracy point lies between $\beta = 1.0$ and $\beta = 4.0$. Because our scale loss can work like regularization of the weight, under the proper choice of β, the network accuracy can be increased.

Table 2: The accuracy and the block usage under various scale condition. The baseline accuracy of ResNet-101 on ImageNet is 76.4. It uses downsample option 'B' in (He et al. 2016). Our best accuracy is achieved at $\mathcal{S} > 0.95$, which is 76.9%.

ImageNet					
	#Blocks	FLOPs(E+10)		Accuracy	
ResNet-72	24.0	1.17		75.8	
ResNet-75	25.0	1.21		75.9	
ResNet-84	28.0	1.34		76.1	
ResNet-101	33.0	1.56		76.4	
$\mathcal{S}$	0.2	0.4	0.6	0.8	1.0
Accuracy	74.0	74.9	75.7	76.4	**76.9**
Block usage	18.78	19.77	22.01	26.94	32.00
FLOPs(E+10)	0.94	0.98	1.08	1.30	1.52

Result on ImageNet

Table 2 is our result on ImageNet (ILSVRC2012). We trained the URNet from ResNet-101 which total 33 blocks. The downsample option 'B'(He et al. 2016) is applied to the (Wu et al. 2018), and we use it too for fair comparision. The result of ResNet-{72, 75, 84, 101} are brought from (Wu et al. 2018). In this experiment, β is set to 4.0. Our method performs better than ResNets with the same amount of computation in all the cases. When $\mathcal{S}$ is about 0.72, our URNet performs equal to ResNet-101 (accuracy: 76.4%) using about 1.24E+10 FLOPs. The accuracy keeps increasing gradually with $\mathcal{S}$, and our best accuracy 76.9% is achieved at $\mathcal{S} > 0.95$. Note that the accuracy of ResNet-101+B/A(rand, train/val) is {26.2%, 49.9%, 64.1%, 71.6%, 76.0%} for each $\mathcal{S}$={0.2, 0.4, 0.6, 0.8, 1.0} (see B/A module in Ablation Study section).



(a) American egret (b) Golden retriever

(c) Peacock (d) Koala

Figure 6: ImageNet samples that activate the blocks differently with an equal scale parameter ($\mathcal{S} = 0.6$). In each object class, the left ones activate 19 blocks of the network, whereas the right ones activate 23 blocks.

(a) Brambling (Night snake) (b) Turtle (Alligator) (c) Mouse (Jack O lantern)

Figure 7: Samples in ImageNet that was correctly classified for large $\mathcal{S}$ (0.8) but misclassified as $\mathcal{S}$ gets reduced (0.6). The misclassified label is written in parentheses. These samples can be regarded as hard samples.

Qualitative Results

Our CGMs not only considering the given $\mathcal{S}$, also consider the input features from the previous layer to decide whether to use the corresponding block or not. In Figure 4, there is green, blue area that represents the block usage is about 0.2∼0.8. These blocks are dynamically opened or closed depending on the input image. These blocks may contain minor but detailed features for hard samples. Figure 6 is the examples of pair of samples that induce the model to activate blocks differently during inference under given $\mathcal{S} = 0.6$. In the Figure, the pair of samples look very different visually. The left ones, which use the minimum number of blocks have very distinctive and remarkable features. Whereas the samples on the right, which need the maximum number of blocks, are hard samples that have too small object (a), too large object (b), too noisy (c), interrupted by other object (d).

Resizable Range

Our URNet can obtain accuracy/FLOPs similar to state-of-the-art compression methods, even though ours has additional characteristics of resizability. As stated previously, we have trained $\mathcal{S}$ with $0.2 \sim 1.0$, but it is hard to satisfy both high performance and large range of $\mathcal{S}$ simultaneously and

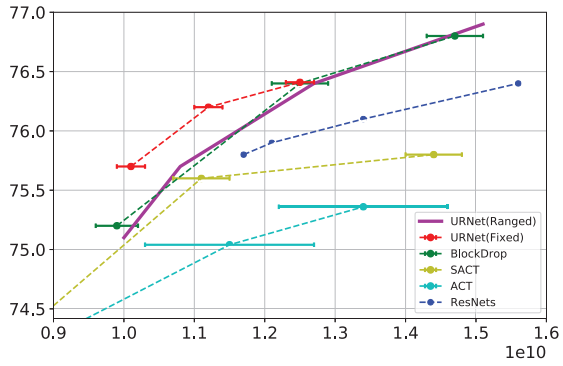


Figure 8: Accuracy vs FLOP. This figure compares URNet(Ranged) and URNet(Fixed) on ImageNet with other methods (Wu et al. 2018; Figurnov et al. 2017). The dot represents one model, and the solid horizontal line represents the standard deviation of one model. URNet(Ranged) represents user resized results at test time by one model. Those of ResNet-{72, 75, 84, 101} and other results are all brought from (Wu et al. 2018).

there exists trade-off between them. In an environment that accepts a more narrow range of $\mathcal{S}$, there is a room to boost performance.

If we train a network with a fixed $\mathcal{S}$ ($\mathcal{S}_{fixed}$), our method can be considered as a static compression method. In this scenario, there is no need to consider the model architecture (number of blocks, kernel size, channel size, etc.) and we just need to set $\mathcal{S}_{fixed}$ as a desirable size.

While the resizable one (URNet(Ranged)) uses various values of $\mathcal{S}$ during training, the fixed scale URNet(Fixed) uses only a small fraction of entire range of $\mathcal{S}$, so there may be the case where only a few blocks are selected to use from the beginning of the training, rather than considering various blocks. To prevent this, the scale parameter $\mathcal{S}_{fixed}$ is initially set to 1, and then gradually reduced to a desirable size. This is called *Scale Annealing* and $\mathcal{S}_{fixed}$ is decayed with the cosine annealing schedule (Loshchilov and Hutter 2016) for specific epochs. In addition, to keep the ability of selectively using blocks, the Gaussian noise is added so $\mathcal{S}_{fixed}$ is sampled from $\mathcal{N}(\mathcal{S}_{fixed}, \sigma^2)$ but restricted not to exceeds 1.

Figure 8 shows the accuracy versus FLOPs of URNet and other compression methods on ImageNet. The solid horizontal line in the figure represents the standard deviation of FLOPs of one model at test time. Note that the URNet(Ranged) is just one model, and can be resized according to user's demand, that others cannot. The URNet(Fixed) is trained with $\mathcal{S}_{fixed} = 0.5$, 0.6 and 0.7, and the Gaussian noise with $\sigma = 0.1$ is added to $\mathcal{S}_{fixed}$ at training time. 5 epochs of scale annealing is applied. Our URNet(Ranged) performs almost equal to BlockDrop, and URNet(Fixed) performs better than that.

Conclusion

We showed that our User-Resizable Residual Networks (URNet) can resize itself as a response to the demand of

a user, at any inference time. Experimental results show that our URNet can change its computational cost without severe accuracy degradation. Unlike other methods, using part of the computational graph according to the pre-defined rules, URNet can determine its computational path by the network itself. Our method can be applied to any ResNet-based network with very little (<0.1%) additional computational burden. Using our method, the user of a network can dynamically balance the number of requests executed per time, by dynamically adjusting the amount of resources per request.

Acknowledgment

This work was supported by Next-Generation Information Computing Development Program through the NRF of Korea (2017M3C4A7077582).

References

Bengio, Y. 2013. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, 1–37. Springer.

Bolukbasi, T.; Wang, J.; Dekel, O.; and Saligrama, V. 2017. Adaptive neural networks for efficient inference. *arXiv preprint arXiv:1702.07811*.

Chang, S.; Yang, J.; Park, S.; and Kwak, N. 2018. Broadcasting convolutional network for visual relational reasoning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 754–769.

Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; and Abbeel, P. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, 2172–2180.

Chen, C.-F.; Fan, Q.; Mallinar, N.; Sercu, T.; and Feris, R. 2018. Big-little net: An efficient multi-scale feature representation for visual and speech recognition. *arXiv preprint arXiv:1807.03848*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. Ieee.

Figurnov, M.; Collins, M. D.; Zhu, Y.; Zhang, L.; Huang, J.; Vetrov, D.; and Salakhutdinov, R. 2017. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1039–1048.

He, Y., and Han, S. 2018. Adc: Automated deep compression and acceleration with reinforcement learning. *arXiv preprint arXiv:1802.03494*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*.

Herbst, N. R.; Kounev, S.; and Reussner, R. H. 2013. Elasticity in cloud computing: What it is, and what it is not. In *ICAC*, volume 13, 23–27.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *stat* 1050:9.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141.

Huang, G., and Chen, D. 2018. Multi-scale dense networks for resource efficient image classification. *ICLR 2018*.

Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. Q. 2016. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 646–661. Springer.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Lin, J.; Rao, Y.; Lu, J.; and Zhou, J. 2017. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, 2181–2191.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. *arXiv preprint arXiv:1312.4400*.

Liu, L., and Deng, J. 2018. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Loshchilov, I., and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*.

Mirza, M., and Osindero, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Odena, A.; Lawson, D.; and Olah, C. 2017. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780*.

Pham, H.; Guan, M.; Zoph, B.; Le, Q.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 4092–4101.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, 3483–3491.

Wu, Z.; Nagarajan, T.; Kumar, A.; Rennie, S.; Davis, L. S.; Grauman, K.; and Feris, R. 2018. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8817–8826.

Yu, R.; Li, A.; Chen, C.-F.; Lai, J.-H.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.-Y.; and Davis, L. S. 2017. Nisp: Pruning networks using neuron importance score propagation. *Preprint at https://arxiv.org/abs/1711.05908*.

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

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.