Dynamic Recursive Neural Network

Qiushan Guo[1], Zhipeng Yu[2], Yichao Wu[2], Ding Liang[2], Haoyu Qin[2], Junjie Yan[2]
[1]Beijing University of Posts and Telecommunications
[2]SenseTime Group Limited
qsguo@bupt.edu.cn
{yuzhipeng,wuyichao,liangding,qinhaoyu,yanjunjie}@sensetime.com

Abstract

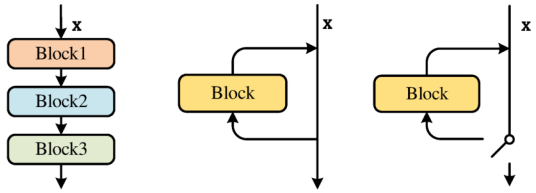
This paper proposes the dynamic recursive neural network (DRNN), which simplifies the duplicated building blocks in deep neural network. Different from forwarding through different blocks sequentially in previous networks, we demonstrate that the DRNN can achieve better performance with fewer blocks by employing block recursively. We further add a gate structure to each block, which can adaptively decide the loop times of recursive blocks to reduce the computational cost. Since the recursive networks are hard to train, we propose the Loopy Variable Batch Normalization (LVBN) to stabilize the volatile gradient. Further, we improve the LVBN to correct statistical bias caused by the gate structure.

Experiments show that the DRNN reduces the parameters and computational cost and while outperforms the original model in term of the accuracy consistently on CIFAR-10 and ImageNet-1k. Lastly we visualize and discuss the relation between image saliency and the number of loop time.

1. Introduction

Deep neural networks (DNNs) have gained huge success in various computer vision tasks in recent years. It is confirmed by many works [13, 34] that deeper and wider models could acquire better results. It is of great challenge to achieve appropriate trade-off between performance and model complexity.

Recently, many efficient network are proposed to overcome this troublesome. Some works concentrate on reforming the internal structure of blocks, like MobileNet [17] obtains novel performance by depthwise-convolution and Inceptions [37] show that block designed delicately can acquire better abilities. Some methods try to optimize blocks [13, 18, 43] by adding reasonable connections among each other. Other methods based on adaptive computation [9, 11, 38, 39, 42] have been proposed as a good alternative option to tackle the model efficiency problem.



(a) (b) (c)

Figure 1: Blocks are executed sequentially in normal networks like (a). Shown in (b), the same block is cyclically executed a fixed times in a recursive network. (c) illustrates that the loop time is dynamically determined by the gate.

According to the above works, we have the following observations: Most existing networks usually contain massive blocks of the identical structure. Some blocks can be skipped to reduce the computational cost. In this work, we attempt to simplify the duplicated blocks and find an effective forward mechanism for different inputs at runtime from novel perspective of block design.

Motivated by the above observation, we propose Dynamic Recursive Neural Network which can reuse blocks dynamically. Fig.1 gives an overview of our approach. In DRNN, feature that is embedded with high-level information can be brought back to refine low-level filters. In this way, the recursive structure makes full use of the parameters. We introduce a gate unit to determine whether to jump out of the loop in advance. It means that different inputs could loop different times in dynamic recursive blocks, which significantly saves computational resources.

The main technical challenge of recursive structure is the problem of gradient explosion and vanish like RNN [2]. Learning an efficient gate unit is difficult while conceptually simple. The gate unit need to make discrete decision and should be end-to-end trained which is empirically optimal. Meanwhile, statistical bias in batch normalization (BN) layer is also an annoying problem caused by different loop time of a recursive block during training.

To tackle with the optimization problem of recursive

structure, we analyze the gradient propagation of recursive block and find that the gradient explosion of a recursive network is caused by BN layer. Therefore, we propose Loopy Variable Batch Normalization to stabilize the gradients and benefit feature re-usage in a more general and easier way than LSTM [10]. In particular, we employ different BN layer for each looping. By applying LVBN, the recursive network can be trained better with fewer parameters.

To incorporate the discrete decisions, we build upon recent work [39] that introduces the Gumbel-Max trick [23] and its differentiable approximations to allow for the propagation of gradient through the discrete decision. Further, the gate unit makes discrete decision deterministically by removing the gumbel noise while maintains the performance.

To deal with the statistical bias, we make an improvement on LVBN, which normalizes all the inputs and updates the population statistics (moving averaged statistics of means and variances across all training images) used in testing time. Some inputs will jump out of loop in advance, while they should contribute to optimizing the gate unit but not to population statistics. So our improved LVBN (I-LVBN) corrects the population statistics by normalizing inputs according to the decision made by the gate unit and updates the population statistics across the inputs which are allowed to pass.

DRNN, combined with the gate unit and I-LVBN, achieves even better performance while accelerating the inference of deep networks. To evaluate DRNN, we use MobileNetV2 [33] and ResNet [13] as the base models on classification and other visual tasks. We approve that, with the progressive strategy designed for recursive network, Dynamic Recursive (DR) ResNet-53 outperforms ResNet-101 while reducing model parameters by 47.0% and computational cost by 35.2%. Further, we study the dynamic recursive behavior of the learned model and reveal the relation between the image saliency and the number of loop time.

Our contributions are listed as follows:

- Presenting a Dynamic Recursive Mechanism to reduce the computational cost.
- Proposing LVBN to stabilize the gradients of recursive networks and make full use of convolutional parameters. Improving LVBN to deal with the statistical bias caused by different loop time of a recursive block during training.
- Model parameters and computational cost can be reduced while obtaining a universal improvement of accuracy.

2. Related Work

Recursive network. Different from the way of sharing weights along the sequence in Recurrent Neural Networks (RNN) [40], recursive network shares weights at every node, which could be considered as a generalization of RNN. [7] tries recursive layers on image recognition but gets worse performance than a single convolution due to overfitting. [36, 30] apply recursive blocks when the input dimension is twice that of output. Studies, incorporating recurrent connections into CNNs, also show superiority in object recognition [27], super-resolution [25] and some other tasks. Recently, based on the studies about recurrent structure and iterative refinement [24, 28], densely connected structure is widely used to obtain more efficient model like DenseNet [18] and CliqueNet [43]. In contrast to these approaches, DRNN behaves as a strategy that can be generally applied in networks with modern block structure. Requiring no complicated connections from former inputs, very deep DRNN performs well on robustness and convergence.

Adaptive computation. The main purpose of adaptive computation is providing "Customized Service" for different inputs to reduce overall inference time, while maintaining or even boosting accuracy. Cascade detectors [8, 41] are early methods that exploit this idea in computer vision, relying on extra prediction modules or handcrafted control strategies. Early prediction models like BranchyNet [38] and Adaptive Computation Time (ACT) [11] adopt branches or halt units to decide whether the model could stop early. Figurnov *et al*. [9] further extend this idea to the spatial domain in ResNet by applying ACT to each spatial position of multiple image blocks. Our approach is closer to the works [39, 42] which add gate unit on every block to determine the execution of block-operation according to current input. Inspired by above methods, a reusable gate unit is designed to reconstruct network structure on the fly conditioned on the input during execution. After every single forward of recursive block, gate unit of recursive block is activated to make a routing choice between going back to block or just passing by and forwarding normally.

Model Compression. Besides critical need of accuracy improvements, reducing storage and inference time also plays an important role in deploying top-performing deep neural networks. Related techniques focus on many fields like distillation [3, 15], filter pruning [5, 31], low-rank factorization [21], quantization [12], compression with structured matrices [4, 35] and network binarization [32]. These works are applied after training the initial networks and usually used as post-processing. DRNN could be trained end-to-end without well-designed training rules.

Other Compact deep nets like SqueezeNet [20] and MobileNet [17, 33] are also end-to-end trainable, but they apply the same computation to all images. Thanks to the loop structure controlled by gate units, DRNN could reuse one block dynamically. In experiments, we prove even compact model like MobileNetV2 could be further improved by applying the dynamic recursive block. Theoretically, recursively used blocks could be further pruned or quantized

by compression method, because internal structure of block has not been changed, and convolution or linear layers still have the potential to be reduced.

3. Dynamic Recursive Neural Network

In this section, we introduce the structure of DRNN, which constructed by the dynamic recursive blocks.

3.1. Naive Recursive Block

Since feature re-usage is able to bring high-level information back to refine low-level filters, the recursive structure could make full use of the parameters as illustrated in Fig. 1b. More precisely, let $\mathbf{x}^i$ be the input of the i^{th} loop of the block, $\mathcal{F}(\mathbf{x}^i)$ be the output and the number of loops is N. We denote $\mathbf{x}^0$ as the input of recursive block, then $\mathbf{x}^N$ is the output of the block.

$$\mathbf{x}^{i+1} = \mathcal{F}(\mathbf{x}^i) \tag{1}$$

To ensure Eq. 1 to be well defined, we require $\mathcal{F}(\mathbf{x}^i)$ and $\mathbf{x}^i$ have the same dimensions. Reusing block simply is considered as Naive Recursive Block. However, the recursive structure gets into trouble during training. We encounter gradient explosion and excessive bias, even though we carefully initialize all the parameters and start with a very small learning rate. According to Eq. 1, we can get the gradient of the parameters in recursive block. Denote $\mathbf{W}_b$ and $\mathcal{L}$ as the parameters of recursive block and the objective function.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_b} &= \frac{\partial \mathcal{L}}{\partial \mathbf{x}^N}\frac{\partial \mathbf{x}^N}{\partial \mathbf{W}_b} + \frac{\partial \mathcal{L}}{\partial \mathbf{x}^N}\frac{\partial \mathbf{x}^N}{\partial \mathbf{x}^{N-1}}\frac{\partial \mathbf{x}^{N-1}}{\partial \mathbf{W}_b} + \dots \\ &+ \frac{\partial \mathcal{L}}{\partial \mathbf{x}^N}\frac{\partial \mathbf{x}^N}{\partial \mathbf{x}^{N-1}}\cdots\frac{\partial \mathbf{x}^2}{\partial \mathbf{x}^1}\frac{\partial \mathbf{x}^1}{\partial \mathbf{W}_b} \\ &= \sum_{i=1}^{N}\frac{\partial \mathcal{L}}{\partial \mathbf{x}^N}\Big(\prod_{j=i}^{N-1}\frac{\partial \mathbf{x}^{j+1}}{\partial \mathbf{x}^j}\Big)\frac{\partial \mathbf{x}^i}{\partial \mathbf{W}_b}\end{aligned} \tag{2}$$

In Eq. 2, $\prod_{j=i}^{N-1}\frac{\partial \mathbf{x}^{j+1}}{\partial \mathbf{x}^j}$ is the main factor of unstable gradient. We find that this instability has nothing to do with convolutional layers. For the convolutional layer, followed with batch normalization layer, back-propagation is unaffected by the scale of its parameters, which has been proved in [22]. The gradient of convolutional layer is stable due to batch normalization. As to activation function, the gradient is 1 or 0 depending on input, so ReLU is not instable in recursive architecture and works fine. So we conjecture that the problem is in the batch normalization layer. In order to solve this problem, we propose to replace batch normalization layers in the block with Loopy Variable Batch Normalization layers, noticing that most of excellent networks have batch normalization layer. That makes the naive recursive network convergent and achieve a good performance.

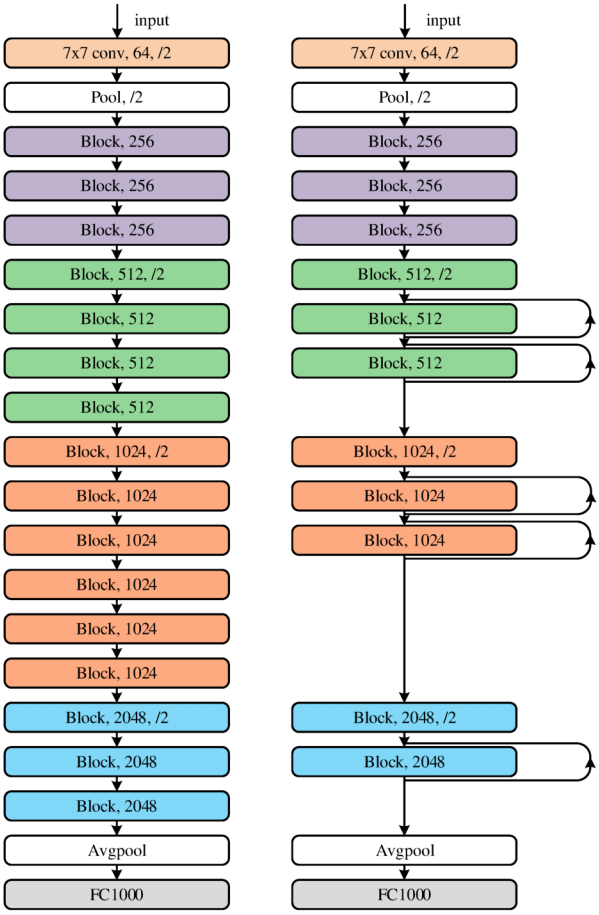


Figure 2: Example network architectures for ImageNet. Left: the ResNet-50 model (25.56M parameters) in [13]. Right: the DR-ResNet 35 model reformed from ResNet-50 (17.61M parameters). More details in Tab. 2

3.2. Loopy Variable Batch Normalize

Training deep neural networks is complicated by internal covariate shift which is proposed in [22]. To reduce internal covariate shift, the BN layer normalizes each scalar feature independently, making it have the mean of zero and the variance of 1, then scales and shifts the normalized value.

$$u^j = \gamma\widehat{x^j} + \beta, \tag{3}$$

γ and β are usually initialized to 1 and 0. According to Eq. 2 and Eq. 3, we use chain rule, as follows:

$$\frac{\partial \mathbf{x}^{j+1}}{\partial \mathbf{x}^j} = \frac{\partial \mathbf{x}^{j+1}}{\partial \mathbf{u}^j}\frac{\partial \mathbf{u}^j}{\partial \widehat{\mathbf{x}^j}}\frac{\partial \widehat{\mathbf{x}^j}}{\partial \mathbf{x}^j} = \gamma\frac{\partial \mathbf{x}^{j+1}}{\partial \mathbf{u}^j}\frac{\partial \widehat{\mathbf{x}^j}}{\partial \mathbf{x}^j}, \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^{N}\frac{\partial \mathcal{L}}{\partial \mathbf{x}^N}\Big(\prod_{j=i}^{N-1}\frac{\partial \mathbf{x}^{j+1}}{\partial \mathbf{x}^j}\Big)\frac{\partial \mathbf{x}^i}{\partial \gamma}, \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^0} = \gamma^N\frac{\partial \mathcal{L}}{\partial \mathbf{x}^N}\prod_{i=0}^{N-1}\frac{\partial \mathbf{x}^{i+1}}{\partial \mathbf{u}^i}\frac{\partial \widehat{\mathbf{x}^i}}{\partial \mathbf{x}^i} \tag{6}$$

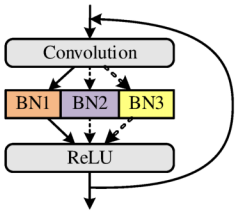


Figure 3: Overview of LVBN when loop time is 3. After the convolution layer, a separated BN layer will be activated each time instead of the same BN layer activated for 3 times. The different style of arrow line represents being into different loop.

We find that the gradient of $\mathbf{x}^0$ is proportional to γ^N, which will cause the unstable gradient propagation. The gradient of γ also contains a term proportional to γ^{N-1}. When γ is greater than 1, it is easy to cause gradient explosion.

During training, a BN layer needs to calculate the mean and variance of each mini-batch and update the population statistics. Whereas, the input features, representing different information, have different means and variances which are then recurrently averaged to the shared population statistics. This will cause incorrect population statistics for each looping. These batch normalization statistics are used during testing which is crucial to the performance.

So we propose LVBN which employs different BN layer for each looping as illustrated in Fig. 3. Training γ and β separately for each looping stabilizes the gradient and enhances the representation ability of the network. Updating the population statistics correctly and separately guarantees the performance during testing.

The cost of these additional BN layers is marginal(less than 1% for bottleneck in ResNets) in the number of parametersand the run-time overhead is also negligible for inference. Based on some excellent networks, we fold some blocks of identical structure which turns the network into a recursive vision (Fig. 2). The results show that LVBN enables the network with recursive block to achieve similar accuracy as the counterpart with the same training procedure but perform better by reducing the learning rate simply (Tab. 3).

Even though an amount of parameters are saved by recursive structure, the inference time hasn't been reduced obviously. We propose a dynamic recursive mechanism to reduce the number of looping and optimization difficulty. Consequently, this mechanism also reduces the cost of computation.

3.3. Dynamic Recursive Mechanism

According to the mismatch of growth rate between parameters and accuracy, we infer that many layers are more concerned with some concepts unrelated to the final performance, which cannot be shared across examples. (e.g. The parameter amount of ResNet-101 is 74% more than ResNet-50, but the performance on ImageNet only increases 1.1%.) Actually, a shallower network is qualified for many examples. For recursive network, it means that fewer loop times could handle the easy examples.

Inspired by the method about gated inference in [39], we propose Dynamic Recursive Mechanism to decide whether to jump out of the loop in advance based on the input. The dynamic recursive block can be defined as

$$\mathbf{x}^{i+1} = \mathcal{G}(\mathbf{x}^i)\mathcal{F}(\mathbf{x}^i) + (1 - \mathcal{G}(\mathbf{x}^i))\mathbf{x}^i \tag{7}$$

where $\mathcal{G}(\mathbf{x}^i) \in \{0, 1\}$ is a gate. The gate has two discrete outputs: execute ($\mathcal{G}(\mathbf{x}^i) = 1$) or skip ($\mathcal{G}(\mathbf{x}^i) = 0$), which is a hard attention mechanism actually.

Our aim is to reduce the average computational cost by skipping some looping. Therefore, we adopt a lightweight and universal design for gate unit shown in Fig. 4. We first apply global average pooling on the input feature map to squeeze global spatial information into a $1 \times 1 \times C$ channel descriptor and then project the feature to unnormalized scores $\mathbf{s}$ for the discrete outputs. We opt to employ two fully connected layers around the non-linearity.

$$\mathbf{s} = \mathbf{W}_2\delta(\mathbf{W}_1\mathbf{x}_\mathbf{p}) \tag{8}$$

where δ refers to the ReLU function, $\mathbf{W}_1 \in \mathbb{R}^{d\times C}$, $\mathbf{W}_2 \in \mathbb{R}^{2\times d}$, $\mathbf{x}_\mathbf{p}$ is the output of global average pooling layer and d is the dimension of the hidden layer. The gate unit constructed by this way adds only a computational overhead of 0.04%.

From the perspective of model design, we need a gate unit can make a discrete decision like $\arg\max$. And following the term of end-to-end training, the gate unit should be differentiable. A straight idea is to choose the maximum of the two scores to decide whether execute or skip during feed-forward. However, this approach is not differentiable. So, we would like choose among the two decisions proportional to their score. For this, we use the Straight-Through Gumbel-Softmax estimator, which is proposed in [1], for propagating gradient through stochastic neurons. The random variable G follows a standard Gumbel distribution if $G = -\log(-\log(U))$ with $U \sim \mathrm{Uniform}(0, 1)$. According to Gumbel-MAX trick, which is a key property of Gumbel distribution, let X be a discrete random variable with $P(X = k) \propto \alpha_k$ and let $\{G_k\}_{k\in\{1,\dots,K\}}$ be an i.i.d sequence of standard Gumbel random variables, then we can sample X by sampling from Gumbel random variables.

$$X = \underset{k\in\{1,\dots,K\}}{\arg\max} (\log \alpha_k + G_k) \tag{9}$$

We set the $\log$ probabilities to the estimated scores, $\log \alpha = \mathbf{s}$. For propagating gradient, we use a continuous relaxation of the Gumbel-Max trick proposed in [23, 29], replacing the

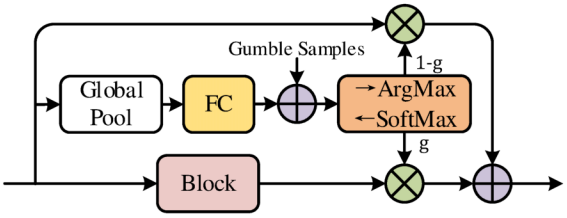


Figure 4: Overview of dynamic recursive mechanism during training. There are three branches. **Top:** shortcut of the original input. **Middle:** gate structure. **Bottom:** sequential layers in block. Global Pooling and FC layers firstly estimate relevance of the block. Then a Gumbel-Max trick is used to choose the output according to the relevance scores. A softmax relaxation is used for backward.

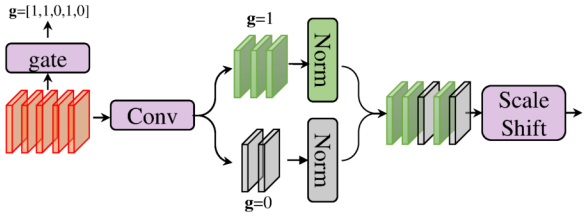


Figure 5: Overview of I-LVBN when batch size is 5. Gate unit first predicts whether the following block is needed conditioned on red input feature maps. Next, the outputs of convolution layer will be divided in to two groups(green and gray) and normalized independently. Whereas, only the population statistics of green group are used in inference. Finally, normalized feature maps will be reindexed, scaled and shifted.

argmax with softmax. The relaxation procedure is summarized by:

$$\mathcal{G}_{relax}(\mathbf{x}) = \begin{cases} \arg\max_{k\in\{0,1\}}(s_k + G_k) \\ \text{softmax}(s_k + G_k) \end{cases} \tag{10}$$

where s_k is the maximum one of $\mathbf{s}$, argmax is used in forward pass and softmax in backward pass. The straight-through estimator performs well with the gumbel sample during training. But the gumbel sample, a noise in fact, is neither necessary nor desirable during inference; we want the output to depend only on the input deterministically. For this, noise is removed in testing.

3.4. Improved Loopy Variable Batch Norm

According to Eq.7, the gradient of $\mathcal{G}(\mathbf{x}^i)$ is related to $\mathcal{F}(\mathbf{x}^i)$, so $\mathcal{F}(\mathbf{x}^i)$ is needed to optimize $\mathcal{G}$ regardless of whether the block is skipped or not. Actually, all the feature maps in a mini-batch need to be input to blocks during training, while LVBN calculates and updates mean and variance of both skipped inputs and executed inputs.

However, the skipped inputs and executed inputs have different mean and variance. This leads to inaccurate BN population statistics in a layer-by-layer propagating manner, which has a negative effect on performance. For this, we propose Improved Loopy Variable Batch Normalization (I-LVBN) to update the statistics correctly as shown in Fig. 5. I-LVBN normalizes the input feature maps respectively by making them have the mean of 0 and the variance of 1, according to the outputs of the gate. Only the mean and variance of executed inputs will be used for inference. The normalization procedure is summarized by:

$$\mathbf{u}_j^i = \gamma^i \widehat{\mathbf{x}_j^i} + \beta^i, \tag{11}$$

where $j \in \{0, 1\}$ is the output of gate and $\mathbf{x}_1^i$ is allowed to pass during inference.

I-LVBN could not only enhance performance for inference by using population statistics based on executed inputs, but also prompt gradient to propagate more precisely during training. This improvement does not lead to increasing parameters or computational overhead.

3.5. Training Loss

The average computational cost of the network depends on how often each block is allowed to be executed. Ideally, for each loop, the block is executed at a certain rate t. It means that when loop twice, the target rate becomes t^2. In this way, the average computational cost will decrease rapidly as the number of loop increases. We estimate the execution rates for each block over mini-batch. Let $\overline{g_j^i}$ denote the ratio of images executed at the i^{th} loop of the j^{th} block. Then, the loss of average computational cost is defined as

$$L_{acc} = \frac{1}{M}\sum_{j=1}^{M}(\sum_{i=1}^{N}\overline{g_j^i} - \sum_{i=1}^{N} t^i)^2 \tag{12}$$

The average computational cost can be easily adjusted by the target execution rate. DRNN are trained end-to-end with a multi-task loss function :

$$L = L_{MC} + L_{acc} \tag{13}$$

where L_{MC} is the standard multi-class logistic loss.

4. Experiments

We perform a series experiments to evaluate our Dynamic Recursive Neural Network on image classification benchmarks.

4.1. Results on CIFAR

Model configurations and training details We first evaluate a range of DR-ResNets architectures on CIFAR-10. Our focus is on the behaviors of our terse and efficient approach, but not on pushing the state-of-the-art results, so we intentionally use the same architectures of block as ResNet. ResNet has a stack of $3n$ blocks on feature maps

Model	Error	Params(10^6)	FLOPs(10^9)
SD-ResNet 110 [19]	5.25	1.7	0.255
Pre-ResNet 110 [14]	6.37	1.7	0.255
AIG-ResNet 110 [39]	5.76	1.78	0.215
RCNN-160 [27]	7.09	1.86	-
ResNet-110 [13]	6.61	1.7	0.255
DR-Res 74 ($l = 2$)	**5.21**	0.92	0.220
DR-Res 66 ($l = 3$)	5.66	0.73	0.214
DR-Res 58 ($l = 4$)	5.79	0.55	0.197
DR-Res 54 ($l = 5$)	5.97	**0.45**	**0.192**
ResNet-56 [13]	6.97	0.85	0.127
DR-Res 40 ($l = 2$)	**6.51**	0.50	0.110
DR-Res 32 ($l = 4$)	6.73	**0.310**	**0.099**
ResNet-20 [13]	8.75	0.27	0.041
DR-Res 16 ($l = 2$)	**8.14**	**0.18**	**0.032**

Table 1: Error rate (%) on CIFAR-10. All of the DR-ResNets outperform their counterpart while using less parameters and computation cost. Our proposed approach for dynamic recursive networks is applicable to both deep network architectures and shallower ones.

of sizes $\{32, 16, 8\}$ respectively, with n blocks for each feature map size. Our DR-ResNet has $\lfloor \frac{(n-1)}{l} \rfloor + 1$ blocks for each feature map size where l is the number of loop. An additional block is for downsampling. Such a design ensures that the maximum computational cost of our network does not exceed its counterpart.

We fix the early blocks up to the first downsampling because the low-level feature maps are not yet distinguish. For the gates, we set the size of hidden state d to 16 and the target executed rate to 0.7.

We follow a similar training scheme as [13] with a weight decay of 0.0005 and momentum of 0.9. The models are trained with a mini-batch size of 256 for 350 epochs. We start with learning rate of 0.1 and divide it by 10 after 150 and 250 epochs. All the images for training are padded with 4 pixels on each side and a 32×32 crop is randomly sampled from the padded image or its horizontal flip.

Results Tab. 1 shows test error, the number of model parameters and floating point operations(multiply-adds) on CIFAR-10 [26]. The first part in the table includes some variant methods based on ResNet and a study that also incorporates recursive structure. The other parts compare our DR-ResNets with ResNets. From the results, we observe that all of the DR-ResNets outperform their counterpart. The more times block loops, the less parameters and computation are need. Whereas, the performances are similar.

Overall, DR-ResNet 54 is able to reduce parameters and computation by **73.53**% and 24.71% while outperforming the ResNet-110. Since the ResNet-110 is overfitted for CIFAR-10, stochastic depth ResNet-110 regularizes by dropping layers and AIG-ResNet 110 applies adaptive inference graph. DR-ResNet 110 outperforms them by reusing the convolutional layer conditioned on the input example. Our proposed method can be applied not only to deep networks, but also to shallower networks like ResNet-20. DR-ResNet 16 can also reduce parameters and computation by 33.3% and 22.0% while outperforming its counterpart.

4.2. Results on ImageNet

In experiments on ImageNet [6], we analyze the effectiveness of our LVBN, dynamic recursive mechanism and Improved LVBN based on ResNet-101 and evaluate a series of network on ImageNet.

Model configurations and training details We build DR-ResNet and DR-MobileNetV2 by inheriting the residual bottleneck and inverted residual block.

Blocks in early stage or not repetitive are fixed. The dynamic recursive block groups are marked bold in Tab. 2, while the downsampling layers stay unchanged. The number of blocks is designed to ensure the maximum computational cost of our network does not exceed its counterpart. The blocks in last group loop twice in DR-ResNets because the last group comprises only three blocks. In DR-MobileNetV2, the loop time of the fourth group is three and the other dynamic recursive blocks loop twice.

For our counterparts of ResNet-101, ResNet-50 and MobileNetV2, the target rate is set to 0.7, 0.8, 0.9. All the gates are initialized at a executed rate of 85% at the beginning of training. The size of the hidden state is 16 for gates in DR-ResNets. Moreover, each gate unit comprises only one fully-connected layer for more compact structure.

We follow the ResNet training procedure, with learning rate starting at 0.1 and decaying by 0.1 every 30 epochs. The weight decay is 0.0001 and momentum is 0.9. All the models are trained for 100 epochs. One exception is that for DR-MobileNetV2 and its counterpart, we start the learning rate at 0.01 and decay it by 0.1 at 200 and 300 epochs. Both are optimized by stochastic gradient descent (SGD) for 400 epochs.

We apply the scale and aspect ratio augmentation as GoogleNet [37] and the photometric distortions method [16]. During test time, the images are rescaled to 256×256 follwed by a 224×224 center crop.

Quantitative comparison Tab. 2 shows top-1 accuracy rate and models' details on ImageNet. From the results we can make the following key observations. DR-ResNet with 65 and 53 layers have better performance than ResNet-101 while using less computational resources. Shallower DRNNs also outperform their counterpart. In particular, DR-ResNet 53 saves 35.2% of computation and 47.0% of parameters. Similarly, DR-ResNet 35 outperforms ResNet-50 while using 33.7% less parameters and 17.8% less average computational cost. For DR-ResNet, increasing loop

Model	Top 1	Top 5	Params(10^6)	FLOPs(10^9)	Blocks
ResNet-101 [13]	77.95	93.86	44.54	7.6	(3,4,23,3)
Stochastic Depth ResNet-101 [19]	77.20	93.56	44.54	7.6	(3,4,23,3)
AIG-ResNet-101 [39]	77.93	93.85	46.23	5.11	(3,4,23,3)
DR-ResNet 65 ($l = 2$)	**78.12**	**93.90**	28.12	5.49	(3,4,**12**,**2**)
DR-ResNet 53 ($l = 3, t = 0.7$)	**77.96**	**93.86**	**23.60**	**4.92**	(3,4,**8**,**2**)
DR-ResNet 53 ($l = 3, t = 0.65$)	77.14	93.52	23.60	4.62	(3,4,**8**,**2**)
DR-ResNet 53 ($l = 3, t = 0.6$)	76.91	93.51	23.60	4.35	(3,4,**8**,**2**)
DR-ResNet 47 ($l = 4$)	77.41	93.54	21.34	4.56	(3,4,**6**,**2**)
DR-ResNet 44 ($l = 5$)	77.27	93.53	20.21	4.25	(3,4,**5**,**2**)
ResNet-50 [13]	76.45	92.90	25.56	3.8	(3,4,6,3)
Stochastic Depth ResNet-50 [19]	72.25	90.86	25.56	3.8	(3,4,6,3)
AIG-ResNet 50 [39]	76.42	93.17	26.56	3.15	(3,4,6,3)
DR-ResNet 35 ($l = 2$)	**76.48**	**92.92**	**17.61**	**3.12**	(3,**3**,**3**,**2**)
MobileNetV2 [33]	71.8	90.27	3.40	0.300	(1,2,3,4,3,3,1)
DR-MobileNetV2	**71.8**	**90.28**	**2.96**	**0.275**	(1,2,**2**,**2**,**2**,**2**,1)

Table 2: Top 1 and Top 5 accuracy rate (%) on ImageNet. Dynamic recursive block groups are bold in table. All the downsampling blocks are not recursive in block groups. The last group blocks loop twice for computational equality in DR-ResNets. The result demonstrates that DR-ResNet is more efficient and also improves overall classification quality. All the MobileNetV2, ResNets and AIG-ResNets are reimplemented with the training procedure in Sec. 4.2.

Model	L	G	I	Top 1	Params (10^6)	FLOPs (10^9)
ResNet-101 [13]				77.95	44.54	7.6
R-ResNet 53				Fail	23.39	7.6
R-ResNet 53	√			77.22	23.39	7.6
R-ResNet 53	√	√		77.52	23.60	4.92
R-ResNet 53	√	√	√	77.96	23.60	4.92
R-ResNet 53*	√			78.38	23.39	7.6

Table 3: The comparative and ablative result of our dynamic recursive network on ImageNet validation set. When Recursive ResNet-53 is trained with smaller learning rate for more epochs, it(R-ResNet 53*) also outperforms ResNet-101.

time performs better than reducing the execution rate. In particular, DR-ResNet 53 with a target rate of 0.65 has a larger expected total loop time than DR-ResNet 47.

For the deeper network, reusing the parameters of convolutional layer first improve accuracy, before increasing the loop time further decrease accuracy insignificantly. This demonstrates that reusing the parameters of convolutional layer is efficient to improve the capacity of network by dynamic recursive mechanism. Further, reusing blocks adaptively is often more effective to save computational resoures compared to reducing blocks of identical structure directly.

As expected, decreasing the target rate reduces computation time. Interestingly, increasing loop times leads to better result than reducing the execution rate. More loop time means that more high-level information is used to refine the low-level filters to get a stronger ability for representation. The recursive structure benefits feature re-usage.

Due to our proposed approach for dynamic recursive networks is general, we also apply dynamic recursive block on MobileNetV2. The result show that the training of dynamic recursive models can be applied to convolution and depthwise-sparable convolution layers in different building blocks.

These results indicate that the parameters of convolutional layer is underused and DRNN is an effective means to adaptively assemble network graph on the fly.

Analysis of dynamic recursive network To understand dynamic recursive network, we conduct ablation experiments to examine how each proposed component affects the final performance. We use R- to indicate naive recursive models. L indicates that we replace BN with LVBN, G stands for gate units and I is the improved LVBN. Each component is an improvement based on the previous one.

From results in Tab. 3, some promising conclusions can be summed up as follows:

- **LVBN is crucial to reuse convolution layers.** Naive recursive network (R-ResNet 53) fails to converge to a good solution and becomes divergent after a few epochs, as illustrated in Fig. 6. LVBN can not only get correct population statistics for each loop but also solve gradient explosion. Utilizing LVBN, we can get a result similar to baseline without changing any hyper-parameters of optimization algorithm. We train a recursive network with LVBN starting with a lower learning rate of 0.01 and weight decay of 0.0005 for 180 epochs. The learning rate is divided by 10 after

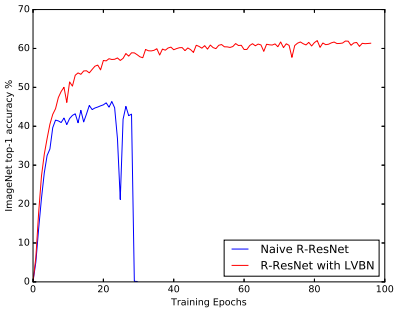


Figure 6: Testing accuracy on validation set of naive R-ResNet 53 and R-RseNet 53 with LVBN. For the naive network, we start with a smaller learning rate of 0.01.

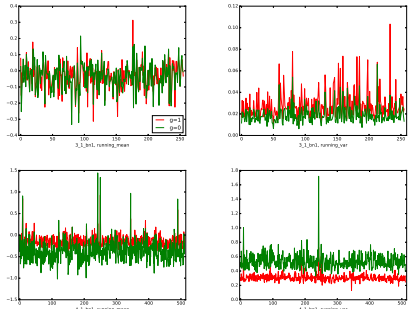


Figure 7: Statistics of the skipped and executed feature maps in DR-ResNet 53. Both shallow and deep blocks are shown. X-axis represents channel index.

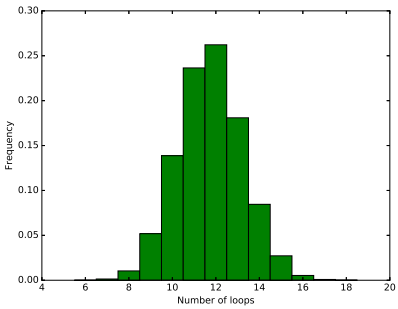


Figure 8: For DR-ResNet 53 on ImageNet, 11.74 out of 23 loops are executed on average. According to the target 0.7, the total loop time has an expect of 11.921.

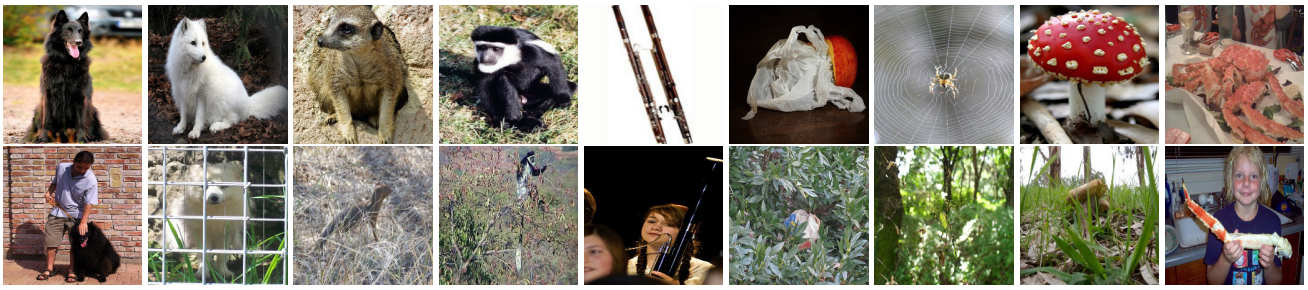


Figure 9: Visualization of easy and hard examples in ImageNet validation set with DR-ResNet 53. The images on the top are easy example (loop less than 9 times) and the bottom ones are hard examples (loop more than 15 times).

100, 130 and 160 epochs. This trick leads to a better top-1 accuracy of 78.38%, the last row of Tab. 3, without adding FLOPs compared to ResNet-101. It shows a trade-off between the size of model and the difficulty of optimizing.

- **Dynamic recursive mechanism is efficient.** Since reusing lots of parameters in network poses extra optimization contrast to unrolled networks, a dynamic recursive network with LVBN is expected to have lower performance. Howerer, the comparison between third and fourth rows in Tab. 3 shows that our dynamic recursive mechanism effectively improves the performance. The prediction cost is reduced by 35.2% while accuracy is increased by 0.4%. The main reason is that our gate unit is efficient and reduces the difficulty of optimizing recursive network by jumping out of loop in advance. The result also indicates that shallower network is easy to optimize and able to tackle most easy examples.
- **Improved LVBN is essential.** It deals with the deviation of population statistics caused by the feature maps which the gates forbid to pass. Each channel of the features maps has different mean and variance. Fig. 7 shows the diverse mean and variance of the different feature maps according to outputs of gates. The results shows that the value discrepancy increases in the deep layer, which indicates that the gate is more confident in deeper layers.

Visualization of loop time Our primary interest lies in understanding the learned gated pattern. Due to the dynamic recursive mechanism, loop time varies across images. Fig. 8 shows the distribution over the total loop times in DR-ResNet 53 are executed on ImageNet validation set. On average 11.74 loops are executed with a standard deviation of 1.50. We collect the easy examples which skip most loops and the hard examples which execute most loops in Fig. 9 for ImageNet validation set. Images in the same column are in the same category. Interestingly, the easy examples are clear and iconic while the hard examples are blurry and occluded, which are even hard for humans to recongize.

5. Conclusion

In this work, we introduce Dynamic Recursive Neural Network that reuses the identical blocks on the fly in a neat way. The usual gradient problem in recursive networks is solved by our LVBN. DRNN also learns to adaptively skip redundant loop based on the input. Further, we correct the population statistics of LVBN which is combined with dynamic recursive mechanism. Experiments on ImageNet and CIFAR show that DRNNs reduce model size and computational cost substantially while outperforming. The proposed DRNN could be further extended to densely-connected or inception-based networks and may help to optimize the learning of long-term dependencies.

References

[1] Y. Bengio. Estimating or propagating gradients through stochastic neurons. *Computer Science*, 2013. 4

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, Mar. 1994. 1

[3] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker. Learning efficient object detection models with knowledge distillation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 742–751. Curran Associates, Inc., 2017. 2

[4] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015. 2

[5] Y. L. Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990. 2

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6

[7] D. Eigen, J. Rolfe, R. Fergus, and Y. Lecun. Understanding deep architectures using a recursive convolutional network. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014. 2

[8] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 2241–2248. IEEE, 2010. 2

[9] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, volume 2, page 7, 2017. 1, 2

[10] A. Graves. *Long Short-Term Memory*. Springer Berlin Heidelberg, 2012. 2

[11] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. 1, 2

[12] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[13] 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*, pages 770–778, 2016. 1, 2, 3, 6, 7

[14] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645, 2016. 6

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. 2

[16] A. G. Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013. 6

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017. 1, 2

[19] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016. 6, 7

[20] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 2

[21] Y. A. Ioannou. Training CNNs with Low-Rank Filters for Efficient Image Classification: ICLR 2016 Poster, May 2016. 2

[22] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015. 3

[23] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. 2016. 2, 4

[24] S. Jastrzebski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio. Residual connections encourage iterative inference. *arXiv preprint arXiv:1710.04773*, 2017. 2

[25] J. Kim, J. Kwon Lee, and K. Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[26] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 6

[27] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2, 6

[28] Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016. 2

[29] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. 2017. 4

[30] P. H. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning (ICML)*, number EPFL-CONF-199822, 2014. 2

[31] A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:1–1, 10 2015. 2

[32] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2

[33] 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*, pages 4510–4520, 2018. 2, 7

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[35] V. Sindhwani, T. Sainath, and S. Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015. 2

[36] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3d object classification. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 656–664. Curran Associates, Inc., 2012. 2

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 6

[38] S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2464–2469. IEEE, 2016. 1, 2

[39] A. Veit and S. Belongie. Convolutional networks with adaptive inference graphs. 2018. 1, 2, 4, 6, 7

[40] O. Vinyals, S. V. Ravuri, and D. Povey. Revisiting recurrent neural networks for robust asr. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4085–4088. IEEE, 2012. 2

[41] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004. 2

[42] X. Wang, F. Yu, Z.-Y. Dou, and J. E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. *arXiv preprint arXiv:1711.09485*, 2017. 1, 2

[43] Y. Yang, Z. Zhong, T. Shen, and Z. Lin. Convolutional neural networks with alternately updated clique. *arXiv preprint arXiv:1802.10419*, 2018. 1, 2