Learning to Search Efficient DenseNet with Layer-wise Pruning

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Deep neural networks have achieved outstanding performance in many real-world applications with the expense of huge computational resources. The DenseNet, one of the recently proposed neural network architecture, has achieved the state-of-the-art performance in many visual tasks. However, it has great redundancy due to the dense connections of the internal structure, which leads to high computational costs in training such dense networks. To address this issue, we design a reinforcement learning framework to search for efficient DenseNet architectures with layer-wise pruning (LWP) for different tasks, while retaining the original advantages of DenseNet, such as feature reuse, short paths, etc. In this framework, an agent evaluates the importance of each connection between any two block layers, and prunes the redundant connections. In addition, a novel reward-shaping trick is introduced to make DenseNet reach a better trade-off between accuracy and float point operations (FLOPs). Our experiments show that DenseNet with LWP is more compact and efficient than existing alternatives.

1 Introduction

Deep neural networks are increasingly used on mobile devices, where computational resources are quite limited(Chollet, 2017; Sandler et al., 2018; Zhang et al., 2017; Ma et al., 2018). Thus, the deep learning community has paid much attention to compressing and accelerating different types of deep neural networks(Gray et al., 2017).

Among recently proposed neural network architectures, DenseNet (Huang et al., 2017b) is one of the most dazzling structures which introduces direct connections between any two layers with the same feature-map size. However, recent extensions of Densenet with careful expert design, such as Multi-scale DenseNet(Huang et al., 2017a) and CondenseNet(Huang et al., 2018), have shown that there exists high redundancy in DenseNet. Neural architecture search (NAS) has been successfully applied to design model architectures for image classification and language models (Liu et al., 2018; Zoph & Le, 2016; Pham et al., 2018; Liu et al., 2017a; Brock et al., 2017). However, none of these NAS methods are efficient for DenseNet due to the dense connectivity between layers. It is thus interesting and important to develop an adaptive strategy for searching an on-demand neural network structure for DenseNet such that it can satisfy both computational budget and inference accuracy requirement.

To this end, we propose a layer-wise pruning method for DenseNet based on reinforcement learning. Our scheme is that an agent learns to prune as many as possible weights and connections while maintaining good accuracy on validation dataset. Our agent learns to output a sequence of actions and receives reward according to the generated network structure on validation datasets. Additionally, our agent automatically generates a curriculum of exploration, enabling effective pruning of neural networks.

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

2 Method

Suppose the DenseNet has L layers, the controller needs to make K (equal to the number of layers in dense blocks) decisions. For layer i, we specify the number of previous layers to be connected in the range between 0 and n_i ($n_i = i$). All possible connections among the DenseNet constitute the action space of the agent. However, the time complexity of traversing the action space is $\mathcal{O}(\prod_{i=1}^{K} 2^{n_i})$, which is NP-hard and unacceptable for DenseNet(Huang et al., 2017b). Fortunately, reinforcement learning is good at solving sequential decision optimization problems and we model the network pruning as a Markov Decision Process(MDP). Since these hierarchical connections have time-series dependencies, it is natural to train LSTM as the controller to simply solve the above-mentioned issue.

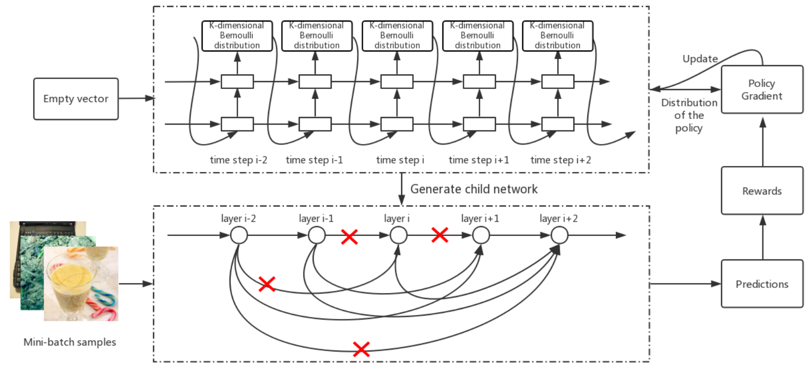

Figure 1: Illustration of our proposed framework. In each iteration, the output of the i-th time step makes keeping or dropping decisions for the i-th layer. All outputs of the LSTM controller generate a child network by sampling from $K \times K$-dimensional Bernoulli distribution. Then, the child network forwards propagation with mini-batch samples and the reward function can be evaluated with the predictions and FLOPs. The controller is optimized with policy gradient.

At the first time step, the LSTM controller receives an empty embedding vector as the input that is regarded as the fixed state $\mathbf{s}$ of the agent, and the output of the previous time step is the input for the next time step. Each output neuron in the LSTM is equipped with $\delta(x) = \frac{1}{1+e^{-x}}$, so that the output $\mathbf{o_i}$ defines a policy $p_{i,\mathbf{a_i}}$ of keeping or dropping connections between the current layer and its previous layers as an n_i-dimensional Bernoulli distribution:

$$\mathbf{o_i} = f(\mathbf{s}; \theta_\mathbf{c}), \qquad p_{i,\mathbf{a_i}} = \prod\nolimits_{j=1}^{n_i} o_{ij}^{a_{ij}} (1 - o_{ij})^{(1-a_{ij})}, \tag{1}$$

where f denotes the controller parameterized with $\theta_\mathbf{c}$. The j-th entry of the output vector $\mathbf{o_i}$, denoted by $o_{ij} \in [0, 1]$, represents the likelihood probability of the corresponding connection between the i-th layer and the j-th layer being kept. The action $\mathbf{a_i} \in \{0, 1\}^{n_i}$ is sampled from Bernoulli($\mathbf{o_i}$). $a_{ij} = 1$ means keeping the connection, otherwise dropping it. Finally, the probability distribution of the whole neural network architecture is formed as:

$$\pi(\mathbf{a}_{1:K} | s; \theta_\mathbf{c}) = \prod\nolimits_{i=1}^{K} p_{i,\mathbf{a}_i} \tag{2}$$

The reward function is designed for each sample and not only considers the prediction correct or not, but also encourages less computation:

$$R(a) = \begin{cases} 1 - \eta^\alpha & \text{if predict correctly} \\ -\gamma & \text{otherwise.} \end{cases} \tag{3}$$

where $\eta = \frac{SUBFLOPs}{FLOPs}$ measures the percentage of float operations utilized. SUBFLOPs, FLOPs represent the float point operations of the child network and vanilla DenseNet, respectively. After obtaining the feedback from the child network, we define the following expected reward:

$$J(\theta_c) = \mathbb{E}_{a \sim \pi_{\theta_\mathbf{c}}} [r(s, a)] \tag{4}$$

To maximize Eq (4) and accelerate policy gradient training over $\theta_\mathbf{c}$, we utilize the advantage actor-critic(A2C) with an estimation of state value function $V(s; \theta_v)$ to derive the gradients of $J(\theta_c)$ as:

$$\nabla_{\theta_c} J(\theta_c) = \sum_a (r(s, a) - V(s; \theta_v)) \, \pi(a|s, \theta_c) \nabla_{\theta_c} \log \pi(a|s, \theta_c) \tag{5}$$

The Eq (5) can be approximated by using the Monte Carlo sampling method:

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{n} \sum_{t=1}^{n} \left(r^{(t)}(s,a) - V(s;\theta_v) \right) \nabla_{\theta_c} \log \pi(a|s,\theta_c) \tag{6}$$

The entire training procedure is divided into three stages: curriculum learning, joint training and training from scratch and they are well defined in Appendix 4.1. Algorithm 1 shows the complete recipe for layer-wise pruning.

3 Experiment and conclusion

The results on CIFAR are reported in Table 1. For CIFAR-10 dataset and the vanilla DenseNet-40-12, our method has reduced the amounts of FLOPs, parameters by nearly 81.4%, 78.2%, respectively and the test error only increase 1.58%. The exponential power α and penalty γ can be tuned to improve the performance. In this experiment, we just modify hyperparameter α from 2 to 3 so that the model complexity($105M$ vs $173M$ FLOPs) is increased while test error rate is reduced to 6.00%.The same law can be observed on the DenseNet-100-12 with LWP. Our algorithm also has advantages on Condensenet (Huang et al., 2018) which needs more expert knowledge and NAS (Zoph & Le, 2016) which takes much search time complexity and needs more parameters but gets higher test error.

We can also observe the results on CIFAR-100 from the Table 1 that the amounts of FLOPs in DenseNet with LWP are just nearly 46.5%, 66.3% of the DenseNet-40-12 and DenseNet-100-12. The compression rates are worse than that for CIFAR-10. This may be caused by the complexity of the CIFAR-100 classification task. The more hard task, the more computation is needed. For

Model	FLOPs	Params	CIFAR-10	CIFAR-100
DenseNet-40-12 (Huang et al., 2017b)(our impl.)	566M	1.10M	5.24	25.09
DenseNet-100-12 (Huang et al., 2017b)(our impl.)	3.63G	7.19M	4.34	20.88
VGG-16-Pruned (Li et al., 2016)	206M	5.40M	6.60	25.28
VGG-19-pruned (Liu et al., 2017b)	195M	2.30M	6.20	-
VGG-19-pruned (Liu et al., 2017b)	250M	5.00M	-	26.52
ResNet-110-pruned (Li et al., 2016)	213M	1.68M	6.45	-
DenseNet-40-pruned (Liu et al., 2017b)	190M	0.66M	5.19	25.28
CondenseNet^{light}-94 (Huang et al., 2018)	122M	0.33M	5.00	24.08
CondenseNet-86 (Huang et al., 2018)	65M	0.52M	5.00	23.64
NAS v2 predicting strides (Zoph & Le, 2016)	-	2.5M	6.01	-
DenseNet-40-12-LWP ($\alpha = 2, \gamma = -0.5$)	105M	0.24M	6.82	-
DenseNet-40-12-LWP ($\alpha = 2, \gamma = -0.5$)	263M	0.66M	-	26.99
DenseNet-40-12-LWP ($\alpha = 3, \gamma = -0.5$)	173M	0.40M	6.00	-
DenseNet-100-12-LWP ($\alpha = 2, \gamma = -0.5$)	716M	1.43M	5.12	-
DenseNet-100-12-LWP ($\alpha = 2, \gamma = -0.5$)	2.42G	5.15M	-	21.14

Table 1: Results on CIFAR. DenseNet-40-12 and DenseNet-100-12 are selected as the backbone CNN on CIFAR dataset and our algorithm is applied to the two models. The FLOPs, parameters and test error of the DenseNet with LWP are compered with the vanilla DenseNet and the neural network architecture with other pruned methods.

ImageNet, although the bottleneck layer and compression ratio are introduced in DenseNet-121-32, the result shows that there is still much redundancy. As observed from Table 2, we can still reduce 54.7% FLOPs and 35.2% parameters of the vanilla DenseNet-121-32 with 1.84% top-1 and 1.28% top-5 test error increasing.

Model	FLOPs	Params	Top-1	Top-5
DenseNet-121-32-BC (Huang et al., 2017b)	5.67G	7.98M	25.35	7.83
DenseNet-121-32-BC-LWP	2.57G	5.17M	27.19	9.11

Table 2: Results on ImageNet. DenseNet-121-32 is selected as the backbone CNN on ImageNet. It can be further compressed even if its parameters are already quite efficient.

In conclusion, we propose an algorithm strategy to search efficient child network of DenseNet with reinforcement learning agent. The LSTM is used as the controller to layer-wise prune the redundancy connections. The whole process is divided into three stages: curriculum learning, joint training and training from scratch. The extensive experiments based on CIFAR and ImageNet show the effectiveness of our method. Analyzing the child network and the filter parameters in every convolution layer prove that our proposed method can learn to search compact and efficient neural network architecture.

References

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

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pp. 1610–02357, 2017.

Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. Technical report, Technical report, OpenAI, 2017.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017a.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017b.

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. *group*, 3(12):11, 2018.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2755–2763. IEEE, 2017b.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2018.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. 2018.

X Zhang, X Zhou, M Lin, and J Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. arxiv 2017. *arXiv preprint arXiv:1707.01083*, 2017.

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

4 Appendix

4.1 Algorithm for layer-wise pruning

Curriculum learning. It is easy to note that the search space scales exponentially with the block layers of DenseNet and there are total $\prod_{i=1}^{K} 2^{n_i}$ keeping/dropping configurations. We use curriculum learning(Bengio, 2013) like BlockDrop(Wu et al., 2018) to solve the problem that policy gradient is sensitive to initialization. For epoch t ($1 \leq t < K$), the LSTM controller only learns the policy of the last t layers and keeps the policy of the remaining $K - t$ layers consistent with the vanilla DenseNet. As $t \geq K$, all block layers are involved in the decision making process.

Joint training. The previous stage just updates parameters θ_c and θ_v. The controller learns to identify connections between two block layers to be kept or dropped. However, it prevents the agent from learning the optimal architecture. Jointly training the DenseNet and controller can be employed as the next stage so that the controller guides the gradients of θ_v to the direction of dropping more connections.

Training from scratch. After joint training, several child networks can be sampled from the policy distribution $\pi(a|s, \theta_c)$ and we select the child network with the highest reward to train from scratch, and thus better experimental results have been produced.

We summarize the entire process in Algorithm 1.

Algorithm 1 The pseudo-code for layer-wise pruning.

```
Input:    Training dataset $\mathcal{D}_t$; Validation dataset $\mathcal{D}_v$; Pretrained DenseNet.
    Initialize the parameters $\theta_c$ of the LSTM controller and $\theta_v$ of the value network randomly.
    Set epochs for curriculum learning, joint training and training from scratch to $M^{cl}$, $M^{jt}$ and $M^{fs}$ respec-
    tively and sample $Z$ child networks.
Output:   The optimal child network
 1: //Curriculum learning
 2: for $t = 1$ to $M^{cl}$ do
 3:     $\mathbf{o} = f(\mathbf{s}; \theta_c)$
 4:     if $t < K - t$ then
 5:         for $i = 1$ to $K - t$ do
 6:             $\mathbf{o}[i, 0 : i] = 1$
 7:             $\mathbf{o}[i, i :] = 0$
 8:         end for
 9:     end if
10:     Sample $\mathbf{a}$ from $Bernoulli(\mathbf{o})$
11:     DenseNet with policy makes predictions on the training dataset $\mathcal{D}_t$
12:     Calculate feedback $R(\mathbf{a})$ with Eq (3)
13:     Update parameters $\theta_c$ and $\theta_v$
14: end for
15: //Joint training
16: for $t = 1$ to $M^{jt}$ do
17:     Simultaneously train DenseNet and the controller
18: end for
19: for $t = 1$ to $Z$ do
20:     Sample a child network from $\pi(\mathbf{a}|\mathbf{s}, \theta_c)$
21:     Execute the child network on the validation dataset $\mathcal{D}_v$
22:     Obtain feedback $R^{(t)}(\mathbf{a})$ with Eq (3)
23: end for
24: Select the child network $\mathcal{N}$ with highest reward
25: //Training from scratch
26: for $t = 1$ to $M^{fs}$ do
27:     Train the child network $\mathcal{N}$ from scratch
28: end for
29: return The optimal child network $\mathcal{N}$
```

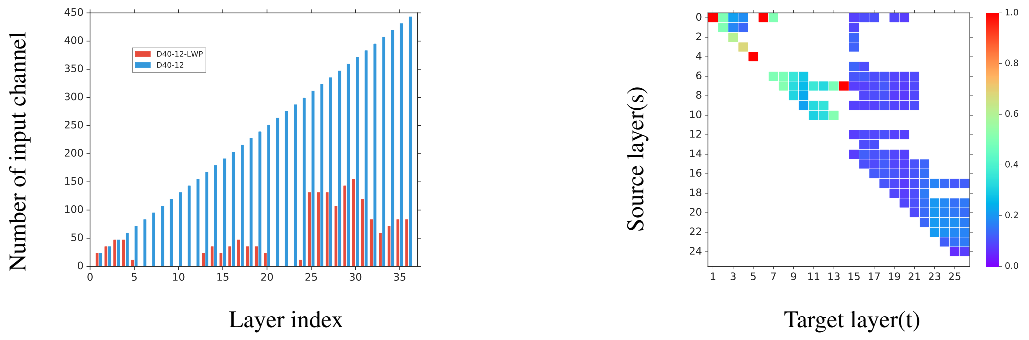

Figure 2: Quantitative results on DenseNet-40-12 wth LWP. $Left$: the number of input channel in vanilla DenseNet-40-12 and the learned child network. $Right$: the connection dependency between any two layers is represented as the average absolute wights of convolution layer.

4.2 Quantitative Results

In this section, we argue that our proposed methods can learn more compact neural network architecture by analyzing the number of input channel in DenseNet layer and the connection dependency between a convolution layer with its preceding layers.

In Figure 2 $left$, the red bar represent the number of input channel in DenseNet-40-12-LWP (D40-12-LWP) and the blue bar represent the number of input channel in vanilla DenseNet. We can observe that the number of input channels grows linearly with the layer index because of the concatenation operation and D40-12-LWP has layer-wise input channels identified by the controller automatically. The input channel is 0 means this layer is dropped so that the block layers is reduced from 36 to 26. The number of connections between a layer with its preceding layers can be obtained from the right panel of Figure 2. In Figure 2 $right$, the x, y axis define the target layer t and source layer s. The small square at position (s, t) represents the connection dependency of target layer t on source layer s. The pixel value of position (s, t) is evaluated with the average absolute filter weights of convolution layers in D40-12-LWP. One small square means one connection and the number of small squares in the vertical direction indicates the number of connections to target layer t.

As reported by the paper DenseNet(Huang et al., 2017b), there are redundant connections because of the low kernel weights on average between some layers. The right panel of Figure 2 obviously shows that the values of these small square connecting the same target layer t are almost equal which means the layer t almost has the same dependency on different preceding layers. Naturally, we can prove that the child network learned from vanilla DenseNet is quite compact and efficient.