Learning to Search Efficient DenseNet with Layer-wise Pruning

Xuanyang Zhang
SMILE Lab, Sch. Computer Science & Engineering
University of Electronic Science & Technology of China
Chengdu, Sichuan, China
xuanyang91.zhang@gmail.com

Hao Liu
Electronic Engineering and Computer Science
University of California, Berkeley
California, USA
lhao499@gmail.com

Zhanxing Zhu
School of Mathematical Science
Peking University
Beijing, China
zhanxing.zhu@pku.edu.cn

Zenglin Xu
[1] *School of Computer Science and Technology*
Harbin Institute of Technology Shenzhen, Guangdong, China
[2] Center for Artificial Intelligence
Peng Cheng Lab, Shenzhen, Guangdong, China
xuzenglin@hit.edu.cn

***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 inference with 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.**

***Index Terms*—DenseNet, Reinforcement learning, Compact neural network**

I. Introduction

Deep neural networks are increasingly used on mobile devices, where computational resources are quite limited [5; 27; 34; 22; 31]. Despite the success of deep neural networks, it is very difficult to make efficient or even real-time inference on low-end devices, due to the intensive computational costs of deep neural networks. Thus, the deep learning community has paid much attention to compressing and accelerating different types of deep neural networks [4].

Among recently proposed neural network architectures, DenseNet [14] is one of the most dazzling structures which introduces direct connections between any two layers with the same feature-map size. It can scale naturally to hundreds of layers, while exhibiting no optimization difficulties. In addition, it achieved state-of-the-art results across several highly competitive datasets. However, recent extensions of DenseNet with careful expert design, such as CondenseNet [13], have shown that there exists high redundancy in DenseNet. Our paper mainly focuses on how to compress and accelerate the DenseNet with less expert knowledge on network design.

A number of approaches have been proposed to compress deep networks. Generally, most approaches can be classified into four categories: parameter pruning and sharing [9; 16; 10; 29], low-rank factorization [30; 32; 8; 23; 28], transferred/compact convolutional filters [6; 33; 17], and knowledge distillation [11]. Unlike these approaches requiring

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

intensive expert experience, automatic neural architecture design has shown its potential in discovering powerful neural network architectures. Neural architecture search (NAS) has been successfully applied to design model architectures for image classification and language models [19; 35; 25; 18; 3]. Most of the above four compression approaches can be conjunction with the neural architecture searched by NAS.

However, NAS algorithms based on reinforcement learning or evolution algorithm [35; 1; 26] are computationally expensive and time consuming. Weight sharing strategy is adopted in recent NAS algorithms[25; 19] to reduce the computation costs, but these approaches have to search the architecture in a shallow network while evaluate in a deeper one. Therefore, these approaches are hard to applied to DenseNet due to its extreme depth and diversity of connections in different blocks. 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 pre-trained 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. As illustrated in Figure 1, our agent learns to output a sequence of actions and receives reward according to the generated network structure on training datasets. Additionally, our agent automatically generates a curriculum of exploration, enabling effective pruning of dense connections.

Extensive experiments on several highly competitive datasets show that our method largely reduces the number of parameters as well as flops, while maintaining or slightly degrading the prediction performance, such that the corresponding network architecture can adaptively achieve a balance between inference accuracy and computational resources.

II. METHOD

We analyze the dense connections of DenseNet in Section II-A, then we model the layer-wise pruning as a Markov decision process (MDP) and design a Long-short term memory(LSTM) [12; 20] controller to generate inference paths in Section II-B. The interaction between the agent (i.e., the LSTM controller) and the environment (i.e., the DenseNet) is described in Figure 2. The reward shaping technique in our method is introduced in Section II-C. Finally, we show the complete training process of LWP in Section II-D.

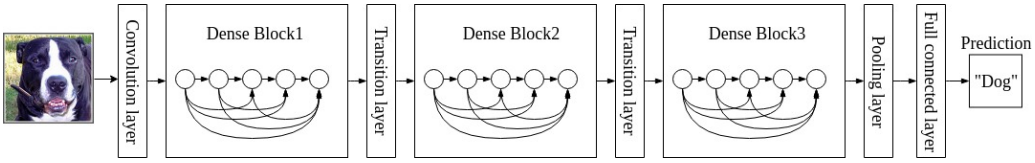
A. Pretrained Dense Convolutional Networks

Vanilla DenseNet consists of four parts: the first convolution layer, multiple dense blocks, transition layers and finally the fully-connected layer. The first convolution layer is only for feature extraction from raw data. As for the multiple dense blocks, each dense block consists of multiple layers. The transition layers are used as down-sampling layers to change the size of feature maps and the last full-connected layer is used for image classification. Obviously, the dense connections are mainly reflected on the dense blocks. Therefore, we study the connection policy for dense layers in this paper.

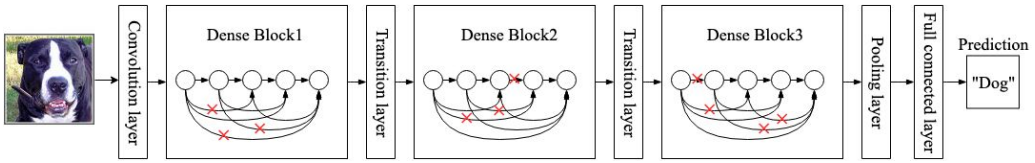
B. Generate Infeience Paths with an LSTM Controller

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 [14]. 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.

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. The activation function of each



(a) Vanilla DenseNet



(b) DenseNet with layer-wise pruning

Fig. 1. An illustration of layer-wise pruning method based on vanilla DenseNet. For one layer, not all connections are required and each layer has its unique connections. After being pruned some dense connections, the network can still predict correctly.

output neuron in the LSTM is $\delta(x) = \frac{1}{1+e^{-x}}$, 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:

$$\mathbf{o_i} = f(\mathbf{s}; \theta_\mathbf{c}) \tag{1}$$

$$p_{i,a_i} = \prod_{j=1}^{n_i} o_{ij}^{a_{ij}} (1 - o_{ij})^{(1-a_{ij})}, \tag{2}$$

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. There are total n_i connections for the i-th layer, but the output dimension of LSTM at each time step is K. To unify the action space dimension and LSTM output dimension, we set both to K and the output of each time step take a $mask \in \{0, 1\}^K$ operation, where the mask numbers from 1-th to n_i-th element are 1 and others are 0. Finally, the probability distribution of the whole neural network architecture is formed as:

$$\pi(\mathbf{a}_{1:K}|s; \theta_\mathbf{c}) = \prod_{i=1}^{K} p_{i,\mathbf{a}_i} \tag{3}$$

C. Reward shaping

Reward shaping is introduced to help the controller make progress to an optimal solution. The reward function is designed for each sample and not only considers the prediction correct or not, but also encourages less computation:

$$R(a) = \begin{cases} 1 - \eta^\alpha & \text{if predict correctly} \\ -\gamma & \text{otherwise.} \end{cases} \tag{4}$$

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. In order to maximize the reward, the prediction needs to be correct and SUBFLOPs should be reduced as much as possible. The trade-off between performance and complexity is mainly controlled by α and γ.

D. Training with Advantage Actor-Critic

After obtaining the feedback from the child network, we define the objective function as the following expected reward:

$$J(\theta_c) = \mathbb{E}_{a \sim \pi_{\theta_\mathbf{c}}}[r(s, a)] \tag{5}$$

To maximize Eq (5) 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:

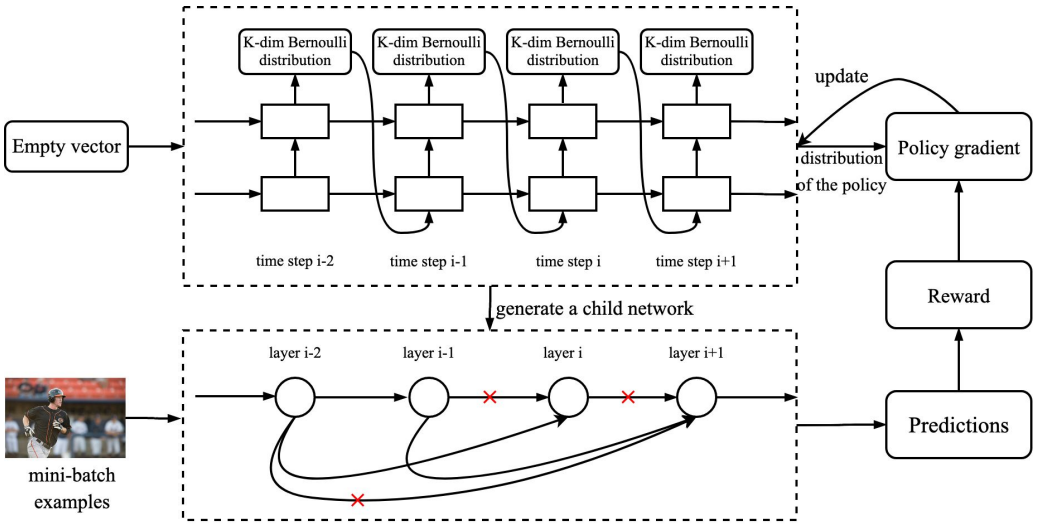


Fig. 2. Illustration of our proposed framework. In each iteration, the output of the i-th time step makes keeping or dropping decisions for the i-th layer. All outputs of the LSTM controller generate a child network by sampling from $K \times K$-dimensional Bernoulli distribution. Then, the child network 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.

$$\nabla_{\theta_c} J(\theta_c) = \sum_a \big[(r(s,a) - V(s;\theta_v))\pi(a|s,\theta_c) * \nabla_{\theta_c} \log \pi(a|s,\theta_c)\big] \tag{6}$$

The Eq (6) can be approximated by using the Monte Carlo sampling method:

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{n} \sum_{t=1}^{n} \big[(r^{(t)}(s,a) - V(s;\theta_v)) * \nabla_{\theta_c} \log \pi(a|s,\theta_c)\big] \tag{7}$$

where n is the batch size. The mini-batch samples share the same child network and perform forward propagation in parallel. Therefore, they have the same policy distribution $\pi(a|s,\theta_c)$ but different $r(s,a)$. We further improve exploration to prevent the policy from converging to suboptimal deterministic policy by adding the entropy of the policy $\pi(a|s,\theta_c)$,$H(\pi(a|s,\theta_c))$ to the objective function. The gradient of the full objective function takes the form:

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{n} \sum_{t=1}^{n} \big[(r^{(t)}(s,a) - V(s,\theta_v)) * \nabla_{\theta_c} \log \pi(a|s,\theta_c) + \beta \nabla_{\theta_c} H(\pi(a|s,\theta_c))\big] \tag{8}$$

As for the value network, we define the loss function as L_v and utilize gradient descent methods to update θ_v:

$$L_v = \frac{1}{n} \sum_{t=1}^{n} \Big(r^{(t)}(s,a) - V(s;\theta_v)\Big)^2 \tag{9}$$

E. Algorithm framework

The entire training procedure is divided into three stages: curriculum learning, joint training and training from scratch.

a) 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 [2] 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 to make process.

b) 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.

c) 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 experiment results have been produced.

III. Experiment

We evaluate the LWP method on three benchmarks: CIFAR-10, CIFAR-100 [15] and ImageNet 2012 [7] for image classification task.

A. Datasets and evaluation metrics

CIFAR-10 and CIFAR-100 consists of 10 and 100 classes images with 32×32 RGB pixels. Both datasets contain $60,000$ images, of which $50,000$ images for training and $10,000$ images for testing. There are total 1.33 million colored images with 1000 visual classes in ImageNet, 1.28 million images for training and $50k$ for validation. The data augmentation schemes the same as DenseNet training [14] were adopted for these three datasets.

B. Results on CIFAR

a) Pretrained DenseNet: For CIFAR datasets, DenseNet-40-12 and DenseNet-100-12 are selected as the backbone CNN. During the training time, the backbone CNN needs to make predictions with dynamic computation paths. In order to make the backbone CNN adjust to our algorithm strategy, we reproduced the DenseNet-40-12 and DenseNet-100-12 on CIFAR based on Pytorch [24] and the results are shown in Table I.

b) Comparisons and analysis: The results on CIFAR are reported in Table I. 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 [13] which needs more expert knowledge and NAS [35] 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 I 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.

C. Results on ImageNet

a) Pretrained DenseNet: We compress the DenseNet-121-32 which has four dense blocks([6, 12, 24, 16]) on ImageNet. The growth rate of DenseNet-121-32 is 32 and this neural network architecture is equipped with bottleneck layers and compression ratio fixed at 0.5 that are designed to improve the model compactness. In the following section, we prove that the model can be further compressed. This model is initialized by loading the checkpoint file of pretrained model from Pytorch.

b) Make comparisons and analysis: 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 II, 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.

TABLE I

RESULTS ON CIFAR. DENSENET-40-12 AND DENSENET-100-12 ARE SELECTED AS THE BACKBONE 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.

Model	FLOPs	Params	CIFAR-10	CIFAR-100
DenseNet-40-12 [14](our impl.)	566M	1.10M	5.24	25.09
DenseNet-100-12 [14](our impl.)	3.63G	7.19M	4.34	20.88
VGG-16-Pruned [16]	206M	5.40M	6.60	25.28
VGG-19-pruned [21]	195M	2.30M	6.20	-
VGG-19-pruned [21]	250M	5.00M	-	26.52
ResNet-110-pruned [16]	213M	1.68M	6.45	-
DenseNet-40-pruned [21]	190M	0.66M	5.19	25.28
CondenseNet^{light}-94 [13]	122M	0.33M	5.00	24.08
CondenseNet-86 [13]	65M	0.52M	5.00	23.64
NAS v2 predicting strides [35]	-	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

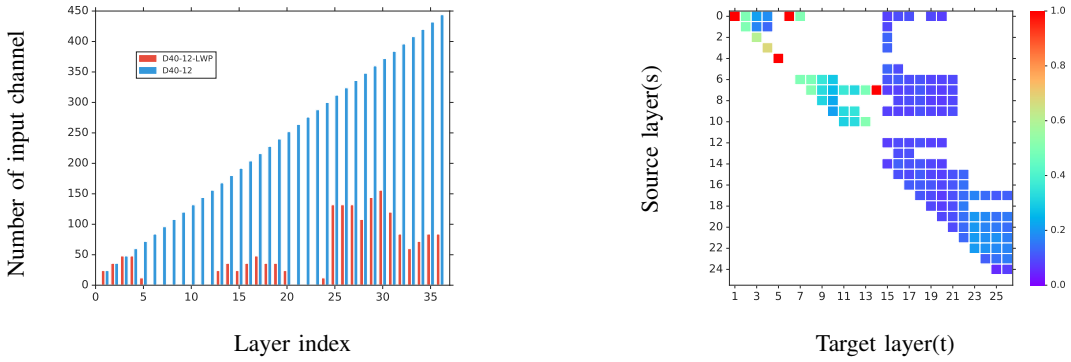


Fig. 3. Quantitative results on DenseNet-40-12 wth LWP. $Left$: the number of input channel in vanilla DenseNet-40-12 and the learned child network. $Right$: the connection dependency between any two layers is represented as the average absolute wights of convolution layer.

TABLE II

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.

Model	FLOPs	Params	Top-1	Top-5
DenseNet-121-32-BC [14]	5.67G	7.98M	25.35	7.83
DenseNet-121-32-BC-LWP	2.57G	5.17M	27.19	9.11

D. 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 convolution layers and the connection dependency between a convolution layer with its preceding layers.

In Figure 3 $left$, 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 channel grows linearly with the layer index because of the feature concatenation 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 3. In Figure 3 $right$, 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.

As DenseNet [14] reported, there are redundant connections because of the low kernel weights on average between some layers. The right panel of Figure 3 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.

IV. 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 proves that our proposed method can learn to search compact and efficient neural network architecture.

Acknowledgement

This work was partially supported by the National Key Research and Development Program of China (No. 2018AAA0100204).

References

[1] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[2] Y. Bengio, "Deep learning of representations: Looking forward," in *International Conference on Statistical Language and Speech Processing*. Springer, 2013, pp. 1–37.

[3] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[4] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[5] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint*, pp. 1610–02 357, 2017.

[6] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*, 2016, pp. 2990–2999.

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

[8] H. Ding, K. Chen, and Q. Huo, "Compressing cnn-dblstm models for ocr with teacher-student learning and tucker decomposition," *Pattern Recognition*, 2019.

[9] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.

[10] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," *group*, vol. 3, no. 12, p. 11, 2018.

[14] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[15] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[17] H. Li, W. Ouyang, and X. Wang, "Multi-bias non-linear activation in deep neural networks," in *International conference on machine learning*, 2016, pp. 221–229.

[18] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *arXiv preprint arXiv:1712.00559*, 2017.

[19] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[20] H. Liu, L. He, H. Bai, B. Dai, K. Bai, and Z. Xu, "Structured inference for recurrent hidden semi-markov model," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018, pp. 2447–2453.

[21] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2755–2763.

[22] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," *arXiv preprint arXiv:1807.11164*, 2018.

[23] Y. Pan, J. Xu, M. Wang, J. Ye, F. Wang, K. Bai, and Z. Xu, "Compressing recurrent neural networks with tensor ring for action recognition," in *The 33rd AAAI Conference on Artificial Intelligence*, 2019, pp. 4683–4690.

[24] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[25] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[26] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," 2017.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *arXiv preprint arXiv:1801.04381*, 2018.

[28] M. Wang, C. Zhang, Y. Pan, J. Xu, and Z. Xu, "Tensor ring restricted boltzmann machines," in *International Joint Conference on Neural Networks*, 2019, pp. 1–8.

[29] L. Wen, X. Zhang, H. Bai, and Z. Xu, "Structured pruning of recurrent neural networks through neuron selection," *Neural Networks*, vol. 123, pp. 134–141, 2020.

[30] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, and T. Chang, "Building fast and compact convolutional neural networks for offline handwritten chinese character recognition," *Pattern Recognition*, vol. 72, pp. 72–81, 2017.

[31] T.-B. Xu, P. Yang, X.-Y. Zhang, and C.-L. Liu, "Lightweightnet: Toward fast and lightweight convolutional neural networks via architecture distillation," *Pattern Recognition*, vol. 88, pp. 272–284, 2019.

[32] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, "Learning compact recurrent neural networks with block-term tensor decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9378–9387.

[33] S. Zhai, Y. Cheng, Z. M. Zhang, and W. Lu, "Doubly convolutional neural networks," in *Advances in neural information processing systems*, 2016, pp. 1082–1090.

[34] 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.

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