DEEP LEARNING BASED METHOD FOR PRUNING DEEP NEURAL NETWORKS

Lianqiang Li[1] *Jie Zhu*[1†] *Ming-Ting Sun*[2†]

[1] Department of Electronic Engineering, Shanghai Jiao Tong University (SJTU), Shanghai, China
[2] Department of Electrical & Computer Engineering, University of Washington, Seattle, WA, USA

ABSTRACT

In order to implement Deep Neural Networks (DNNs) into mobile devices, network pruning has been widely explored for lightening the complexity of Deep Neural Networks in terms of computational cost and parameter-storage load. In this paper, we propose a novel filter-level pruning method which utilizes a deep learning method to pursue compact DNNs. Specifically, we use a DNN model to extract features from the filters at first. Then, we employ a clustering algorithm to force the extracted features roll into different groups. By mapping the clustering results to the filters, we get the "similarity" relationships among the filters. At last, we keep the filter which is closest to the centroid in each cluster, prune out the others, and retrain the pruned DNN model. Compared with previous methods that employ heuristic ways on filters directly or selecting shallow features from filters manually, our method takes advantages of the deep learning method which can represent the raw filters in a more precise way. Experimental results show that our method outperforms several state-of-the-art pruning methods with negligible accuracy loss.

Index Terms— Network pruning, filter-level, deep learning

1. INTRODUCTION

DNNs have gained impressive success in a wide range of artificial intelligence tasks such as image classification [1], natural language processing [2], and video compression [3]. The superior performance of DNNs usually comes from the deeper and wider architectures, which leads to high computational cost and high memory footprint. As a result, these over-parameterized DNN models are very hard to be deployed on mobile devices where computational capability and memory storage are limited. In order to alleviate this issue, many research efforts have been conducted, including low rank approximation [4], parameter quantization [5], and network pruning [6].

Among the existing methods, network pruning is one of the most promising techniques to reduce the complexity of DNNs. Han et al. [6] and Guo et al. [7] introduced weight-level pruning methods. Their methods could prune a large number of parameters. However, the weight-level pruning methods may result in irregular structures, which cannot be easily translated into computational speedups as they are not friendly to the existing BLAS libraries and hardware. In contrast, pruning filters is a more efficient way to discard parameters. Hao et al. [8] and Luo et al. [9] proposed to prune unimportant filters according to some specific "importance" criteria. For example, [8] argued that the filters with lower absolute weight sums have less "importance" and can be pruned away. Another line of filter-level pruning works in [10, 11] assumed that filters have different functions in capturing different features. Therefore, they pruned filters according to their "similarity". Although these filter-level pruning methods have made some progress in accelerating DNNs, there are still rooms to be explored. Pruning filters is challenging because removing filters in one layer means reducing a lot of input for the next layer. Hereafter, a reasonable criterion for selecting pruning candidates is very important.

In this work, we put forward a deep learning based filter-level pruning method which explores the "similarity" among filters to select the pruning candidates. When designing the "similarity" metric, we raise a question - can we learn a good "similarity" measure? Inspiring by the excellent representation capability of deep learning technique, we use a DNN model to depict the filters and then evaluate their "similarity". Concretely, our method consists of three main modules. Firstly, we treat the filters as input and feed them into a DNN model, then extract the output of the last fully-connected layer as the representations for the filters. In fact, the output of the last fully-connected layer is a feature vector for each filter. Secondly, we utilize the k-means [12] algorithm to cluster the vectors into groups, and map the clustering results to the filters to obtain the "similarity" relationships among them. Finally, we retain the filter which is closest to the centroid in each cluster, and prune out the others. Reducing similar filters may degrade the performance, as a result, a retraining process is added to compensate the accuracy loss after pruning. After the whole process, the resulting DNN is much more compact with less computational cost and parameter-storage requirement. Compared with previous works, the proposed method employs a DNN model to capture the structures and patterns

† Jie Zhu & Ming-Ting Sun are the corresponding authors. E-mail addresses: zhujie@sjtu.edu.cn, mts@uw.edu

978-1-5386-9214-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMEW.2019.00-68

of the filters instead of carrying out heuristic ways on filters directly or selecting shallow features from filters manually. Experiments on several benchmark datasets and different network architectures show that our method is better than state-of-the-arts. To the best of our knowledge, it is the first time to use the deep learning technique to assist pruning DNNs.

The rest of this paper is organized as follows. Section 2 introduces several related works on network pruning. Section 3 illustrates the proposed method in detail. In Section 4, we provide the experimental results to show the effectiveness of our method. Finally, we conclude and present our future research direction in Section 5.

2. RELATED WORKS

Network pruning methods have turned out to be effective for lightening the DNNs. We review some of them which are more related to this paper in the following.

Han et al. [6] employed the L1/L2-norm regularizations to remove the weights which are smaller than a certain threshold. Their method can reduce significant number of parameters. However, the pruned parameters are mainly from the fully-connected layers where the computational cost is low. In addition, the weight-level pruning method incurs irregular connectivity patterns which demand dedicated software or hardware for implementation. So, their pruning method cannot transfer accelerations for DNNs directly.

Hao et al. [8] proposed a similar criterion as shown in [6]. They considered the filters with less absolute weight sums are less important and could be discarded. The idea behind this method is that if the weight sum of a filter is small then the activations of the filter are small. These small activations could have little influence on the output and hence the filter could be pruned away. This method not only works well in pruning DNNs but also has the ability to solve the problems resulted from weight-level pruning methods. However, Li et al. [10] argued that the idea in [8] is not very convincing, since every filter in the convolutional layers has its own function in extracting features. A filter with small absolute weight sum can only suggest that it is different from the filters with large ones. As a result, they employed the k-means++ algorithm to explore the "similarity" of the filters and only retain some representative ones. Their performance is comparable to that in [6, 7, 8, 9].

Although the above methods work well to some extent, there are still rooms for improvements. In this paper, similar to the works in [10, 11], we merge the filters with close correlations into one instead of removing filters according to some "importance" criteria. The main rationale behind merging the correlated filters is that different filters have different functions in capturing features. In the ideal scenario, the correlations among the filters are 1 and removing the same filters will not lead to changes in the final performance. However, in practice, none of the filters are fully correlated, and thus, merging them into one filter will affect the accuracy. As a result, it is a significant challenge to define the "similarity" relationships among the filters for pruning.

3. THE PROPOSED APPROACH

We introduce the proposed method for pruning DNNs in this section. Firstly, we introduce the symbol and notations. The deep neural networks can be parameterized by a series of 4-D tensors, $\boldsymbol{W}^i \in R^{N_i \times C_i \times h_i \times w_i}$ $(1 \leq i \leq L)$, where N_i is the number of filters, C_i is the number of channels, h_i and w_i are the spatial height and width of filters, respectively. L is the number of the layers. $\boldsymbol{W}^i$ consists of a bunch of 3-D filters $\boldsymbol{W}^i_n$ $(1 \leq n \leq N_i)$.

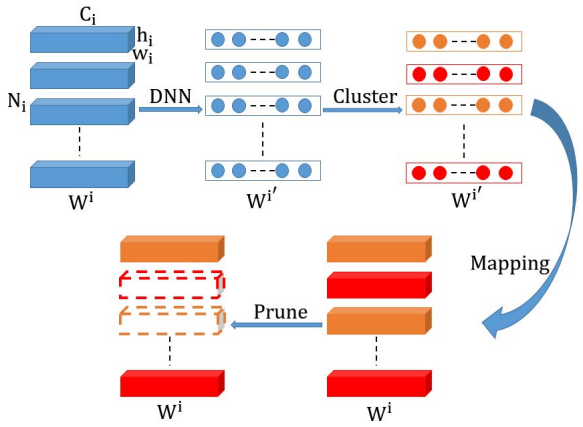


Fig. 1. The proposed pruning method for layer i in a DNN

How to decide whether a filter, $\boldsymbol{W}^i_n$ $(1 \leq n \leq N_i)$, should be pruned or not is an open problem. There are several methods as we described in Section2. In a word, the core idea is to find the filters that could be removed without hurting the original performance severely. Our intuition is that the filters are responsible for capturing features from the input and they have different functions. Thus, we can represent those filters more tersely by using the k-means algorithm. After forcing the filters to roll into k_i groups in layer i, we can replace each filter by the centroid of its corresponding cluster. Actually, pruning $(N_i - k_i)$ filters in layer i can reduce $(N_i - k_i)/N_i$ of the computational cost for both layers i and $i+1$ as the pruned filers' corresponding feature maps will also be removed.

One possible issue is that the DNNs are characterized the N_i distinctive filters in layer i. Some of them may have similar functions in extracting features but are far from each other in distance using a particular "similarity" metric. Actually, it is hard to define a "similarity" metric that works well for all situations. To this end, we utilize a deep learning based method to draw meaningful representations for the filters. Theoretically, a DNN with three layers trained with Back-Propagation (BP) is enough to approximate any nonlinear functions with arbitrary precision [13]. In order to obtain

precise representations, we employ VGG-16 [14] here. VGG-16 is originally designed for ImageNet classification [15]. It has very powerful ability in extracting features and has been used in many scenarios [16, 17]. In detail, we extract the output of the last fully-connected layer in VGG-16, $\boldsymbol{W}^{i'}$, as the representations for the filters. $\boldsymbol{W}^{i'}$ disentangle the filter shapes, norms and other shallow information. Then, we carry out the k-means algorithm on $\boldsymbol{W}^{i'}$ rather than $\boldsymbol{W}^{i}$. After that, we map the clustering results to the filters to obtain the "similarity" relationships among them. At last, we keep the filter which is closest to the centroid in each cluster, prune out the others and retrain the pruned DNN model. The proposed algorithm is summarized in Algorithm. 1.

Algorithm 1 Deep Learning Based Method for Pruning Deep Neural Networks

Input: a DNN model A for pruning, another DNN model B for representing the filters in model A, pruning ratio P_i (percentage of filters removed) for layer i

Output: The pruned DNN model A

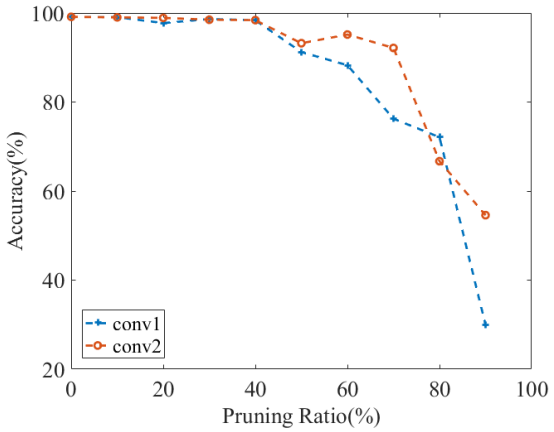
1: $i = 1$
2: **repeat**
3: Forward model B by $\boldsymbol{W}^{i}$ in model A, get the output, $\boldsymbol{W}^{i'}$, as the representation of $\boldsymbol{W}^{i}$
4: Initialize the number of clusters $k_i = N_i \times P_i$
5: Cluster the filters of $\boldsymbol{W}^{i'}$ into k_i clusters with the k-means algorithm
6: Keep the filter which is the most closed to centroid for every cluster in $\boldsymbol{W}^{i}$, prune out the others
7: Retrain the pruned model A as training process to make up for the performance loss
8: $i = i + 1$
9: **until** i is greater to L

It is worth noting that our method could also be used in fully-connected layers because we can treat a neuron as a filter with 1×1 spatial size. However, we consider that using the strategy in [6] in fully-connected layers may be better. There are two main reasons. On one hand, the parameters in the fully-connected layers are directly responsible for feature weighting. We can get higher probability outputs if we keep the larger weights in the fully-connected layers. On the other hand, as we mentioned above, compared to the computational load in convolutional layers, the computational load resulted from the irregular weight matrices in fully-connected layers is negligible.

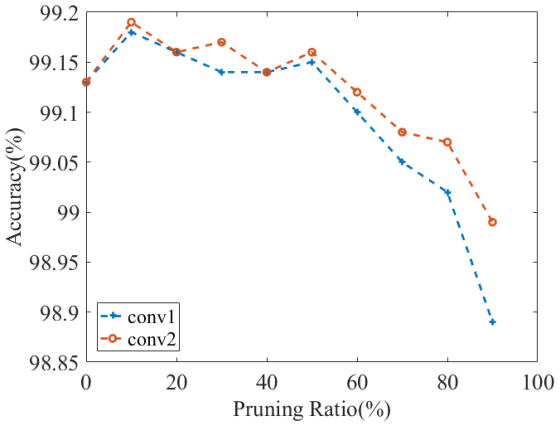
4. EXPERIMENTS AND RESULTS

In this section, we use Caffe [18] to demonstrate the effectiveness of the proposed method on two benchmark datasets, MNIST and CIFAR-10. Two classical convolutional neural networks, Lenet-5 [19] and VGG-16 are used to validate on the two datasets, respectively. For better understanding, our method is compared with [8], [10], and a straightforward filter-level pruning strategy: random selection.

In normal cases, researchers have to carry out considerable experiments (i.e. $\prod_1^L N_i$) to assess the tradeoffs between pruning performance and pruning ratio to get the optimal results. In this paper, we follow the strategy in [6, 8, 10] to prune and analyze the sensitivity of every convolutional layer independently, and decide the pruning ratio. As for the pruning ratio for fully-connected layers, we set it be 90% in our all experiments. The retraining process in our experiments is same as the training process. Empirically, the proposed method is quite robust to the hyper-parameter settings across different DNNs and datasets.



(a) Accuracy v.s. pruning ratio for each single layer of LeNet-5 on MNIST before retraining



(b) Accuracy v.s. pruning ratio for each single layer of LeNet-5 on MNIST after retraining

Fig. 2. Pruning sensitivity of convolutional layers in Lenet-5

4.1. Lenet-5 on MNIST

Lenet-5 consists of 2 convolutional and 2 fully-connected layers, which achieves 99.13% accuracy performance on MNIST.

Fig. 2 presents the pruning sensitivity of convolutional layers in Lenet-5. We can see from Fig. 2(a) that both the first convolutional layer and the second convolutional layer can hold the accuracy performance when pruning ratio is low even there is no retraining process. However, with the increase of pruning ratio, the performance of the both layers decreases inevitably, and the performance of the first layer is slightly worse than that in the second one. It is also clear from Fig. 2(b) that the performance of both layers rises after the retraining process, which means the proposed method works well. Furthermore, the performance of the first layer is still worse than that in the second one. Since our goal is pruning away as many parameters as possible while retain the accuracy performance, we set the pruning ratio 80% for the first convolutional layer and the pruning ratio 90% for the second convolutional layer. In this setting, the loss of accuracy for the two layers is small.

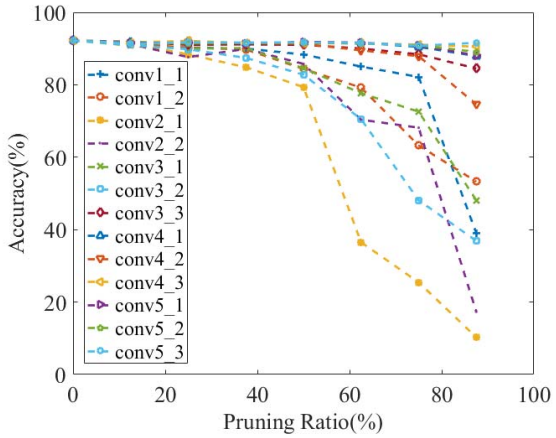
Table 1. The performance of different pruning methods on LeNet-5 under the pruning ratio 89.99% and the pruned FLOPs 94.33%

	Accuracy(%)			
	ours	k-means++	weight sum	random
conv1	**99.07**	99.02	99.04	98.79
conv2	**98.88**	98.61	98.51	98.34
fc1	**98.84**	98.52	98.35	98.15
fc2	**98.82**	98.47	98.31	98.11
final	**98.82**	98.47	98.31	98.11

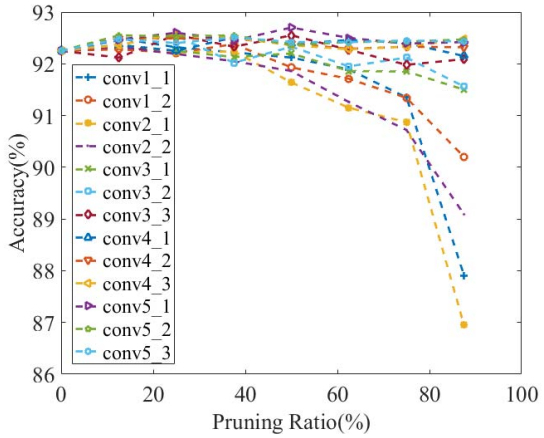
Table. 1 shows the accuracy performance under the same compression situation. It is obvious that the proposed method always outperforms the others. Its final accuracy performance is 99.82% with only 10.01% parameters (Paras.) and 5.67% floating point operations (FLOPs) remained. As for the others, the performance of k-means++ and weight sum is close to each other while the performance of random selection is the worst one.

4.2. VGG-16 on CIFAR-10

VGG-16 has 13 convolutional layers and 3 fully-connected layers. In our experiments, the architecture of the original VGG-16 is kept except the first two fully-connected layers are redesigned as 512 neurons. The network is trained from scratch and gains 92.25% accuracy performance on CIFAR-10. Unlike Lenet-5 on MNIST in which most of the parameters lie in fully-connected layers while the computational cost mainly consists in the convolutional layers, the parameters and computational cost of VGG-16 on CIFAR-10 mainly come from the convolutional layers. Thus, it may be more convincing to validate the effectiveness of our method.



(a) Accuracy v.s. pruning ratio for each single layer of VGG-16 on CIFAR-10 before retraining



(b) Accuracy v.s. pruning ratio for each single layer of VGG-16 on CIFAR-10 after retraining

Fig. 3. Pruning sensitivity of convolutional layers in VGG-16

Fig. 3 presents the pruning sensitivity of the convolutional layers in VGG-16. We can find from Fig. 3(a) that higher layers (layer conv4_x, conv5_x) contain more unnecessary filters than middle layers (layer conv3_x) and lower layers (layer conv1_x, conv2_x) as removing 87.5% of the filters in higher layers has no negative effect on the accuracy performance. We can observe from Fig. 3(b) that the performance of all layers has been compensated to some extent after the retraining process. We can also observe that the performance of the lower layers drops more significantly. All of these suggest that the pruning sensitivity of the low layers is higher than the others. We suspect this sensitivity is due to the lower layers are closed to the input and thus their functions in extracting features are not as powerful as the top layers. Taking into account the degree of decline in accuracy and not introducing

layer-wise hyper-parameters, we use 87.5% pruning ration for the higher layers, 50% pruning ratio for the middle layers and 12.5% pruning ratio for the lower layers.

Table 2. The performance of different pruning methods on VGG-16 under the pruning ratio 86.49% and the pruned FLOPs 74.85%

	Accuracy(%)			
	ours	k-means++	weight sum	random
conv1_1	**92.49**	92.35	92.39	92.22
conv1_2	**92.53**	92.51	92.50	92.47
conv2_1	**92.41**	92.24	92.22	92.19
conv2_2	**92.37**	92.07	92.11	92.05
conv3_1	**92.31**	91.92	91.91	91.89
conv3_2	**92.18**	91.82	91.83	91.66
conv3_3	**92.05**	91.67	91.66	91.42
conv4_1	**92.23**	91.73	91.77	91.55
conv4_2	**92.21**	91.78	91.75	91.52
conv4_3	**92.21**	91.72	91.73	91.47
conv5_1	**92.16**	91.64	91.65	91.36
conv5_2	**92.15**	91.58	91.51	91.27
conv5_3	**92.06**	91.32	91.26	91.13
fc6	**92.04**	91.25	91.14	91.09
fc7	**92.04**	91.24	91.15	91.07
final	**92.04**	91.24	91.15	91.07

Table. 2 shows the accuracy performance under the same compression situation. We can find that the proposed method is still the best one. It just employs 13.51% parameters and 25.15% FLOPs to get 92.04% accuracy performance, which is very closed to the baseline accuracy. The superior performance can be attributed as follows. Our proposed method employs a DNN model to extract high-level features from filters with no regard of their shapes, norms or other shallow information, and we enforce similar filters be assigned into the same group. The pruning of the filters does not affect the accuracy much since they are in the same group and are capturing similar features. The accuracy of k-means++ is closer to ours, which also demonstrates the idea of exploiting the "similarity" among filters is more reasonable. As for the weight sum method, we find that the performance is similar to that in random selection. Since, there is no direct correlations between weights magnitude and the loss function, the method of weight sum may sometimes remove some important filters. Compared to the others, the performance of random selection is a bit of unsatisfactory as its strategy is less logical. We also observe the phenomenon which has been acknowledged in [6, 8, 10] that low pruning ratio in the bottom layers will result in some small benefits. However, it is surprising to find that the performance of all four methods is improved when we prune the layer conv4_x, we believe both phenomena are due to the pruning process helps VGG-16 on CIFAR-10 to avoid overfitting.

5. CONCLUSIONS

In this paper, we propose a deep learning based method for accelerating DNNs. The proposed method utilizes a DNN to interpret the filters as feature vectors with more complexity and abstraction. Besides, it uses the k-means algorithm to explore the "similarity" relationships among them. The effectiveness of the proposed method is demonstrated by comparing with state-of-the-art methods on classical DNNs.

In the future, we will try to employ the unsupervised learning methods to retrain the filters in the DNN model which is responsible for extracting features (i.e. VGG-16 in this paper) to make our proposed method even better. In addition, we would also like to conduct more research on the effect of pruning on other applications except image classification and implement the research into mobile devices.

6. ACKNOWLEDGMENT

This work is supported by the National Key Research Project of China under Grant No. 2017YFF0210903, the National Natural Science Foundation of China under Grant Nos. 61371147 and 11433002 and China Scholarship Council.

7. REFERENCES

[1] Wei Li, Guodong Wu, Fan Zhang, and Qian Du, "Hyperspectral image classification using deep pixel-pair features," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 844–853, 2017.

[2] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun, "Adversarial ranking for language generation," in *Advances in Neural Information Processing Systems*, 2017, pp. 3155–3165.

[3] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Zhiyong Gao, and Ming-Ting Sun, "Deep kalman filtering network for video compression artifact reduction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 568–584.

[4] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao, "On compressing deep models by low rank and sparse decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7370–7379.

[5] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[6] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient

neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[7] Yiwen Guo, Anbang Yao, and Yurong Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

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

[9] Jian-Hao Luo and Jianxin Wu, "An entropy-based pruning method for cnn compression," *arXiv preprint arXiv:1706.05791*, 2017.

[10] Lianqiang Li, Yuhui Xu, and Jie Zhu, "Filter level pruning based on similar feature extraction for convolutional neural networks," *IEICE Transactions on Information and Systems*, vol. 101, no. 4, pp. 1203–1206, 2018.

[11] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin, "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions," *arXiv preprint arXiv:1806.09228*, 2018.

[12] James MacQueen et al., "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Oakland, CA, USA, 1967, vol. 1, pp. 281–297.

[13] James NK Liu, Yanxing Hu, Jane Jia You, and Pak Wai Chan, "Deep neural network based feature representation for weather forecasting," in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014, p. 1.

[14] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[16] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos, "Learning complexity-aware cascades for deep pedestrian detection," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[17] Jiaolong Yang, Peiran Ren, Dongqing Zhang, Dong Chen, Fang Wen, Hongdong Li, and Gang Hua, "Neural aggregation network for video face recognition.," in *CVPR*, 2017, vol. 4, p. 7.

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.