Pruning Convolutional Filters using Distinctiveness Metric

Weiming Ren

Research School of Computer Science, Australian National University
u7025570@anu.edu.au

Abstract. Convolutional neural networks (CNNs) have achieved great success in various computer vision tasks such as object recognition, object detection, and semantic segmentation. Nowadays, people have been building CNNs with deeper layers and more hidden neurons in order to generate richer features and get more accurate results. However, this often leads to over-parameterized networks, causing the deployment cost of the CNNs to be very expensive. We propose distinctiveness-based pruning, a structured pruning method that can effectively remove redundant filters from the convolutional layers and reduce the parameter number of the network. We devise a classification task over a synthetic vehicle dataset and test the pruning method on a classic CNN architecture VGG-11. Our experimental results show that distinctiveness-based pruning can reduce up to 56% of the parameters from VGG-11 while preserving almost the same classification accuracy as the original network after finetuning.

Keywords: Distinctiveness-based pruning · Structured pruning · Convolutional neural network · Vehicle classification

1 Introduction

With the advancement of modern computing hardware, the general research direction of convolutional neural networks (CNNs) in recent years has been to build deeper networks with more convolutional layers. While the capacity and representation ability of deep convolutional networks have been continuously increasing, these cumbersome networks also introduce massive deployment overheads, making them hard or even impossible to be deployed to devices where the computational budget is limited. CNN with millions of parameters requires large storage space and long inference time that are often impossible to meet on devices like embedding sensors or smartphones.

Pruning convolutional neural networks have been proved to be an effective method for solving the above problems. Successful implementation of a network pruning technique will result in efficient and compact neural networks without too much performance damage. The idea of pruning a neural network dates back to the early 1990s when LeCun et al. proposed Optimal Brain Damage [7] that uses the Hessian matrix of the loss function to prune unimportant weights. Gedeon et al. [3] in 1992 used distinctiveness-based pruning to progressively prune an auto-associative network [6] for image compression and got decent decompressed image quality. In the context of convolutional neural networks, we have also witnessed a lot of promising pruning methods proposed in recent years. For example, Li et al. [8] used the L_1-norm of a convolutional filter to prune unimportant filters from the convolutional layers. Liu et al. [10] proposed the network slimming method, which leverages the batch normalization parameter γ as a scaling factor for determining the importance of a convolutional filter and pruning filters with small γ values.

Our work aims at extending the distinctiveness-based pruning method proposed by Gedeon et al. [3] to the context of convolutional neural networks. We present a structured pruning method that can effectively remove filters with duplicated functionalities from the convolutional layers. To prove the effectiveness of our method, we explore a vehicle classification problem using VGG-11 [11] as the feature extraction backbone. We train this model using a large-scale synthetic vehicle dataset and then prune a certain number of filters from each convolutional layer in the network based on their distinctiveness. Our experiments show that distinctiveness-based pruning removes a significant number of parameters from the model and preserves a decent validation/test accuracy.

2 Related Works

2.1 Convolutional Neural Network

A convolutional neural network (CNN) is a type of artificial neural network (ANN) that uses the convolution operator to perform computation. It was initially developed to reduce the computation cost of the standard fully

connected neural networks and has been successful in solving problems like object detection, image classification, semantic segmentation, etc.

When processing high dimensional data like high-resolution images, traditional fully connected networks assign a weight to every pixel on the image for each hidden neuron, which results in the network containing too many parameters and consuming enormous memory space during computation. CNN reduces the parameter number of a network using two methods. Firstly, it assumes that the input data has the characteristic of local correlation. In other words, for every input data point, it only correlates to some data points that are spatially close to this point. CNN uses a square window centered around this data point to indicate this correlation, called the local receptive field. Every output neuron only connects to the data points inside the local receptive field, which greatly reduces computation. Secondly, CNN applies weight sharing for all hidden neurons. No matter where the local receptive field is, the CNN only uses a single set of weights to perform computation. The output of the neuron is then the weighted sum of all the input data points, which is known as the cross-correlation operation in image processing. CNN was inspired by the biological process of the neurons in the animal visual cortex and is highly efficient and effective.

2.2 CNN pruning

CNN pruning aims at removing redundant parameters from the convolutional neural networks without dramatically hurting the network performance. In general, modern CNN pruning methods can be classified into two categories: *unstructured pruning* and *structured pruning* [1]. Figure 1 illustrates the difference between unstructured and structured pruning methods.

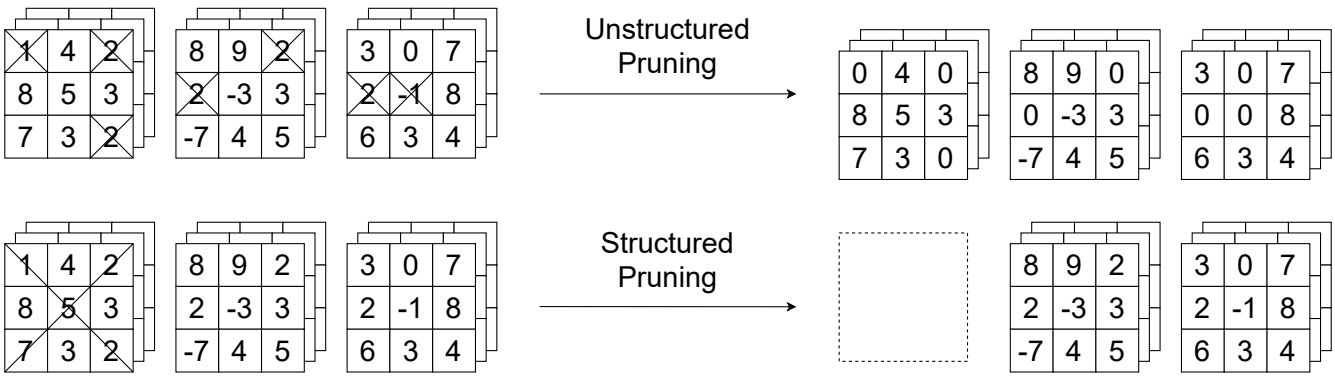

Fig. 1. Illustration of the unstructured pruning and structured pruning method for convolutional neural networks. The granularity of these two pruning schemes are different: unstructured pruning aims at pruning weights, while structured pruning focuses on pruning filters.

For unstructured pruning, the aim is to prune individual weights from the convolutional filters. Because weights are the smallest computing units in convolutional neural networks, this method can guarantee minimum performance loss. However, unstructured pruning cannot reduce the size of the network as it only introduces sparse connections in CNNs but does not change the dimension of the convolutional filters. Unless the computing hardware is designed for sparse matrix calculation, the speed of the network will not increase after unstructured pruning.

Structured pruning aims at pruning filters from the convolutional layers. The removal of the redundant filters indicates that the total parameter number of the network can also be reduced. Therefore, this technique helps reduce the size of the network and increases the network inference speed. Because of this characteristic, our work focuses on extending distinctiveness-based pruning as a structured pruning method.

3 Distinctiveness-based Pruning for Convolutional Neural Networks

We now formally introduce the distinctiveness-based pruning method for structured CNN pruning. Section 3.1 first introduces the initial distinctiveness-based method for fully connected networks. Section 3.2 and 3.3 then introduces how we extend this method to convolutional neural networks.

3.1 Distinctiveness-based Pruning

Distinctiveness-based pruning was initially developed for purning fully connected networks. As the name implies, distinctiveness-based pruning determines the importance of a hidden neuron in the network by inspecting the distinctiveness of its activations. That is, for hidden neuron i in the neural network, we compute the output activation values over a set of inputs and gather these activations to form an activation vector $\boldsymbol{a_i}$:

$$\boldsymbol{a_i} = \left[a^i_{d_1}, a^i_{d_2}, a^i_{d_3}, ..., a^i_{d_M}\right] \tag{1}$$

Where d_m is the m-th data in the input dataset. After we get the activation vector for every neuron in one hidden layer, we then compute the angle of the activation vectors for every hidden neuron pair (i, j):

$$\theta(i,j) = \arccos\left(\frac{\boldsymbol{a_i} \cdot \boldsymbol{a_j}}{||\boldsymbol{a_i}|| \cdot ||\boldsymbol{a_j}||}\right) \tag{2}$$

This angle represents the similarity between neuron i and j. A small θ indicates that the two activation vectors are very similar, which means that the two neurons have almost identical outputs. Therefore, we can prune one of the neurons from the network without causing significant damage to the network performance. When pruning neurons from the network, we add the weights of the pruned neuron to the remained neuron so that the network does not require retraining. Conversely, a θ that is close to 180° means that the two neurons have complementary output values. Therefore, their impact on subsequent parts of the network will be offset. In this case, both neurons can be removed from the network.

3.2 Pruning Filters using Distinctiveness

To extend the pruning method to convolutional neural networks, we need to define the distinctiveness of individual filters in a convolutional layer. Assume a convolutional layer changes the dimension of the input feature map from $\mathbb{R}^{m \times h_1 \times w_1}$ to $\mathbb{R}^{n \times h_2 \times w_2}$. That is, this convolutional layer uses n convolutional filters $f \in \mathbb{R}^{m \times k \times k}$ and each filter outputs a feature map $\mathcal{M} \in \mathbb{R}^{1 \times h_2 \times w_2}$. Similar to the original pruning method, we define the activation vector of a convolutional filter f as:

$$\boldsymbol{a_f} = concat(flatten(\mathcal{M}_1), flatten(\mathcal{M}_2), flatten(\mathcal{M}_3), ..., flatten(\mathcal{M}_B)) \tag{3}$$

Where $flatten(\mathcal{M})$ refers to the operation of reshaping a feature map $\mathcal{M} \in \mathbb{R}^{1 \times h_2 \times w_2}$ to a 1-D vector $\boldsymbol{m} \in \mathbb{R}^{(h_2 \times w_2)}$. $\{\mathcal{M}_1, \mathcal{M}_2, ...\mathcal{M}_B\}$ indicates the collection of output feature maps generated from the filter f on an input image batch. $concat(\boldsymbol{m_1}, \boldsymbol{m_2}, ..., \boldsymbol{m_B})$ means the concatenation operation between vector $\{\boldsymbol{m_1}, \boldsymbol{m_2}, ..., \boldsymbol{m_B}\}$. Therefore, the activation vector $\boldsymbol{a_f}$ for filter f has the dimension of $\mathbb{R}^{(B \times h_2 \times w_2)}$.

After computing the activation vector for each filter in a convolutional layer, we then perform the same angle computation as Equation (2) to derive the pairwise cosine similarity between the filters. Similar to Section 3.1, for a pair of convolutional filters that has a small θ value, we prune one of the filters from the convolutional layer. For θ that is close to 180°, both filters can be removed from the layer. Note that in CNN pruning we do not add the weights from the pruned filter to the remained filter to preserve network performance. Instead, we perform an extra finetuning step after pruning to recover the performance loss for the pruned network.

3.3 Structured CNN Pruning

After determined which filter should be pruned from the convolutional layer, the next step is to perform structured pruning and construct the pruned network. Note that pruning a filter from a convolutional layer also requires modifying the channels of all the filters in the subsequent layer. As shown in Figure 2, the pruning algorithm removes the red filter from the first convolutional layer, which results in the removal of the red channel in the intermediate feature map. Therefore, for every filter in the next convolutional layer, its corresponding channel (in red) should also be removed to avoid the dimension mismatch between the intermediate feature map and the filters. Overall, the structured pruning algorithm can be summarized into four steps:

1) Determine which filter should be pruned from the convolutional layer based on its distinctiveness;
2) Prune filters from the convolutional layer. This changes the dimension of the convolutional kernel from $\mathbb{R}^{n \times m \times k \times k}$ to $\mathbb{R}^{(n-p) \times m \times k \times k}$, where p is the total number of the pruned filters. m, n are the numbers of the channels for the input and output feature map for this convolutional layer.

3) Modify the filters in the next convolutional layer to avoid dimension mismatch. This step changes the size of the filters in the next layer from $\mathbb{R}^{n\times k\times k}$ to $\mathbb{R}^{(n-p)\times k\times k}$.
4) For all the convolutional layers, repeat steps 1-3 until the algorithm reaches the last convolutional layer. If there are fully connected layers after the final convolutional layer, the hidden neuron number of the first fully connected layer should also be modified according to how the output feature map is reshaped and fed into the fully connected network. The hidden neurons that correspond to the elements in the output feature map that are pruned from the network should be removed.

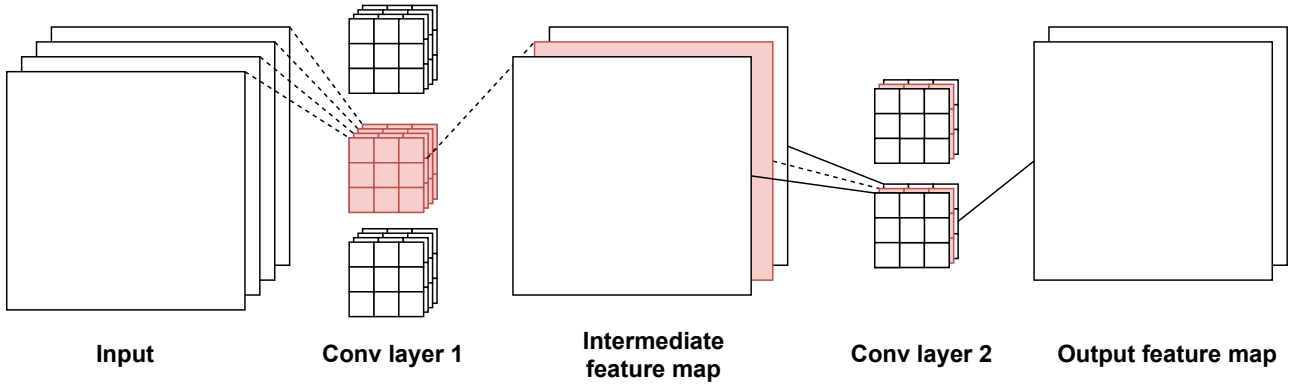

Fig. 2. Illustration of the methodology for structured pruning method. Pruning convolutional layer 1 results in change of channel dimension for the intermediate feature map and the filters in convolutional layer 2.

4 Experimental Methodology

To study the effectiveness of distinctiveness-based pruning, we devise a vehicle classification problem using a large-scale synthetic vehicle dataset VehicleX [13]. We employ the VGG-11 feature extraction backbone [11] as our base network architecture and test the pruning performance on this network. The below sections describe our experiment settings in detail.

4.1 Dataset

The VehicleX dataset is a large-scale synthetic vehicle dataset rendered using Unity 3D. It was initially built for approximating and augmenting the real-world vehicle re-identification datasets in order to enhance the vehicle re-id performance. VehicleX contains a total number of 1,362 different vehicles, 45,438 training images, 14,396 validation images and 15,142 test images. Each image also comes with a detailed list of labels, including camera calibration data, vehicle color, vehicle orientation, vehicle type, etc.

4.2 Experiment Design

Model input and training target. We preprocess VehicleX by normalizing the RGB value of the input images to $[0, 1]$ for floating point representations. This is done by dividing the input images by 255. Our training target is set to be the vehicle type of the training inputs. The VehicleX dataset contains a total of 11 vehicle types, which are sedan, SUV, van, hatchback, MPV, pickup, bus, truck, estate, sportscar, and RV [13].

Learning objective. As we are dealing with a multi-class classification problem, we use the Cross-Entropy loss as the loss function of our model. The Cross-Entropy loss can be expressed using the following equation:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N}\sum_{c=1}^{C} gt_i^c \log(p_i^c) \tag{4}$$

Where gt_i^c and p_i^c correspond to the binary ground truth label and predicted probability of class c for the i-th input, respectively. N is the total number of training inputs and C is the total number of classes. In this case, C is equal to 11.

Network architecture. We employ the VGG-11 [11] feature extraction backbone as the network architecture for the classification task. Our network contains eight 3×3 convolutional layers and three fully connected layers. Each convolutional layer is followed by an ReLU activation function and a max pooling layer is inserted after convolutional layer 1, 2, 4, 6, 8. After VGG-11 outputs the final feature map, we reshape this feature map into a 1-D vector and feed this vector to a fully connected classification network. This network has two hidden layers, both of which contain 4096 hidden neurons. The output of our network is a vector with 11 entries, each of them represents the probability of the input image belonging to a certain vehicle type.The detailed illustration of our network architecture is shown in Figure 3.

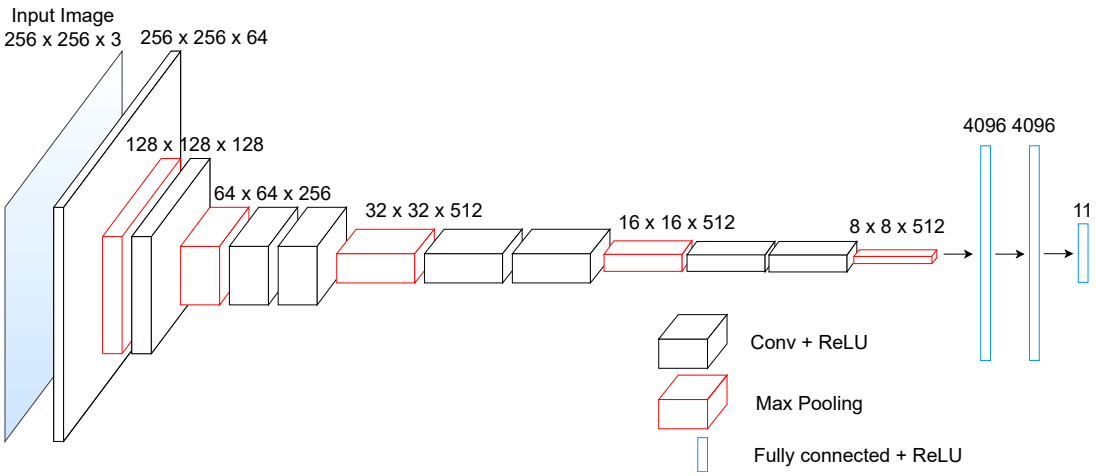

Fig. 3. Illustration of the proposed network architecture including the VGG-11 feature extration backbone and a fully connected classification network.

Pruning configurations. After the network has been trained, we employ distinctiveness-based pruning to prune all eight convolutional layers to remove unnecessary filters. When performing network pruning, we use the raw output feature map from each convolutional filter to compute the activation vector. The angle θ between two filters should be between $[0, \pi]$. We then set up two angle thresholds to prune similar and complementary neurons from the hidden layer.

Experiment setups. We design a set of experimental models to study the effectiveness of distinctiveness-based pruning and finetuning. These experimental models are:

1) A baseline network that is unpruned and acts as a control group;
2) A network that is pruned using distinctiveness-based pruning;
3) A network that is pruned and finetuned to recover its performance damage.

4.3 Implementation Details

We implement all the models using Pytorch 1.8.1 and Python 3.7.6. Our network is trained using the standard stochastic gradient descent (SGD) algorithm with a learning rate of 0.01 and a training batch size of 32. The network is trained for a total of 30 epochs. For distinctiveness-based pruning, we use eight images randomly selected from the VehicleX training dataset to generate activation vectors for the convolutional filters. We define the lower (similar) threshold of the activation vector angle to 30° and the upper (complementary) threshold to 150°. After training and pruning the networks, we then finetune the pruned network using SGD for another 30 epochs. The learning rate and batch size for finetuning are also set as 0.01 and 64.

5 Experimental Results

5.1 Model Evaluation Results

We test the performance of the baseline model and the pruned models and report their test/validation accuracy. We also compute the total parameter number, computational complexity, storage space requirement and running

speed for each model to evaluate the compression performance of the pruned models. Table 1 and Table 2 shows the detailed experimental results.

Table 1. Model scale statistics. *Computational Complexity* and *Parameter Count* correspond the total multiply-add operation and parameter number for each experiment model, respectively. These two statistics are evaluated using the Pytorch Flops counter (`ptflops`, `https://github.com/sovrasov/flops-counter.pytorch`).

Model	Computational Complexity	Pruned (%)	Parameter Count	Pruned (%)	Storage Space	Pruned (%)
Baseline	9.96 GMac	0%	160.27 M	0%	611 MB	0%
Finetuned	9.86 GMac	1.0%	69.88 M	56.4%	266 MB	56.5%

Table 2. Model evaluation results showing the relative error or increase of the pruned models compare with the baseline model. Our test environment for model speed evaluation is equipped with a GTX 1660Ti GPU, an Intel Core i7-9750 CPU and 16GB of RAMs. *Finetuned* and *Pruned* refers to the pruned models with or without finetuning.

Model	Validation Accuracy	Error (%)	Test Accuracy	Error (%)	Running Speed (FPS)	Increased By (%)
Baseline	0.8533	0%	0.8453	0%	569	0%
Pruned	0.1958	77.1%	0.2007	76.3%	647	13.7%
Finetuned	0.8381	1.8%	0.8362	1.1%	663	16.5%

5.2 Result Analysis

Effectiveness of distinctiveness-based pruning. As shown in Table 1, the pruning method is able to remove 56% of the parameters from the network, which shrinks the storage requirement of the model from 611 MB to only 266 MB. This result indicates that pruning is an effective method for convolutional neural network slimming. Note that for VGG-11, more than 90% of the parameters are from the fully connected network, in which the first hidden layer contains 84% of the parameters. Therefore, we find that pruning the last convolutional layer is especially effective for reducing the size of the network as pruning this layer vastly reduces the dimension of the input vector to the fully connected network.

According to Table 2, the test model after pruning-finetuning only suffers from less than 2% validation/test accuracy drops, while the model speed after pruning is able to increase more than 10%. This result demonstrates that our pruning method is very effective and robust to the VGG-11 network. Also, comparing the pruned and finetuned model, we observe that although pruning causes serious performance damage to the original model, finetuning can recover most of the performance loss, and the finetuned model can perform almost as good as the baseline model. This observation indicates that the pruning-finetuning pipeline is effective for both reducing model size and retaining model performance.

Impact of the pruning dataset input. We observe from Table 2 that although the pruning method can remove more than half of the parameters from the model, the computational complexity of the pruned model stays almost the same as the original baseline model. This result indicates that a large number of the parameters in VGG-11 do not occupy too many computations during network inferencing, and thus our pruning method removes them from the network without losing model performance. Our experiments also show that the computational complexity (total multiply-add operations) for a neural network is a good indicator of the network capacity and representation ability. In fact, existing literature has already used this metric to evaluate the cost of a network. For example, Standley et al. [12] define the Standard Network Time (SNT), which is the total multiply-adds for Xception [2]. They then use this unit to regulate the computational cost of different models to ensure fairness in their experiments.

6 Conclusion and Future Work

We have shown from the experiments that distinctiveness-based pruning is a robust network compression method and can effectively remove redundant convolutional filters with little network performance loss. Our experiments present a recommended pruning configuration that uses eight randomly chosen images from the VehicleX training

set to generate the activation vectors of the filters and then prunes the network using a similar threshold of 30° and a complementary threshold of 150°. The pruned network is then finetuned to recover from performance damage. This method is able to prune 56% of the parameters from the original network while only causing less than 2% network performance damage.

For future works, we aim at testing the pruning method on more CNN architectures to further validate its effectiveness and utility. Specifically, as the VGG-11 architecture in our experiments only consists of standard convolutions, it is necessary to develop pruning schemes for more complex architectures such as residual blocks [4], lateral connections [9], and depthwise separable convolutions [2, 5]. Also, due to the hardware limitation of the experiment environment, we were only able to perform distinctiveness-based pruning using eight input images. In the future, we may try pruning the network using more input data to check the stability of the pruning method and determine the best input data size.

References

1. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC) **13**(3), 1–18 (2017)
2. Chollet, F.: Xception: deep learning with depthwise separable convolutions (2016). arXiv preprint arXiv:1610.02357 (2016)
3. Gedeon, T., Harris, D.: Progressive image compression. In: [Proceedings 1992] IJCNN International Joint Conference on Neural Networks. vol. 4, pp. 403–407. IEEE (1992)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arxiv 2015. arXiv preprint arXiv:1512.03385 (2015)
5. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
6. Kramer, M.A.: Autoassociative neural networks. Computers & chemical engineering **16**(4), 313–328 (1992)
7. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990)
8. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
9. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2117–2125 (2017)
10. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2736–2744 (2017)
11. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
12. Standley, T., Zamir, A., Chen, D., Guibas, L., Malik, J., Savarese, S.: Which tasks should be learned together in multi-task learning? In: International Conference on Machine Learning. pp. 9120–9132. PMLR (2020)
13. Yao, Y., Zheng, L., Yang, X., Naphade, M., Gedeon, T.: Simulating content consistent vehicle datasets with attribute descent. In: ECCV (2020)