Evaluating Extended Pruning on Object Detection Neural Networks

Simon O'Keeffe
Department of Electronic Engineering
Maynooth University
Maynooth, Ireland
simon.okeeffe.2010@mumail.ie

Rudi Villing
Department of Electronic Engineering
Maynooth University
Maynooth, Ireland
rudi.villing@mu.ie

***Abstract*— CNNs are the state-of-the-art for many computer vision problems, including object detection. However, reducing the computational complexity of a CNN is a key prerequisite to deploying state-of-the-art deep learning networks in many low power embedded real-time robotic applications. Pruning has been shown to be an effective method to reduce the computational complexity of a Convolutional Neural Network (CNN) while maintaining accuracy. In the literature, accuracy lost through pruning is recovered with extended fine-tuning of the pruned network at the end of the pruning procedure, but further pruning is not conducted after extended fine-tuning. In this work we modify the pruning procedure to incorporate extended fine-tuning at intervals during the procedure to maintain network accuracy while pruning further than would otherwise be possible. We evaluate this procedure on a small scale custom object detection dataset and the more challenging standard PASCAL VOC dataset. On the former the new procedure achieves a 19.6× reduction in FLOPS for a drop of only 0.4% mean Average Precision (mAP) while the latter achieves only a 1.8× reduction in FLOPS for a drop of 0.8% mAP. The results indicate differing levels of parameter redundancy in the initial networks.**

Keywords—Convolutional Neural Networks, pruning, real time, embedded, multi-class object detection

I. INTRODUCTION

For many applications including autonomous driving, pedestrian detection, and service robots, multiclass object detection needs to be performed accurately in real-time, on platforms that may have low computational power. State-of-the-art deep learning object detection methods [1, 2] currently require significant computing power and memory to run in real time. These methods utilise large Convolutional Neural Networks (CNNs) which require GPUs to achieve real time performance. To deploy deep learning methods to embedded devices, CNNs need to be smaller and faster. Smaller networks require less memory to store the network parameters which is valuable because embedded devices may not have enough memory to store larger networks [3]. Faster networks require fewer floating point operations, which may allow them to run in real time. Pruning is one method that has been shown to be effective at making CNNs smaller and faster [4].

CNNs have been shown to have a lot of redundancy [5]. Pruning takes advantage of this redundancy by removing weights and connections in a network. In CNNs, pruning may be used to remove entire kernels from the network. By removing entire kernels what remains is still a standard CNN and no specialised computational techniques are needed. This effectively reduces the computational load of the network. To maintain network accuracy the network requires further training after pruning, known as *fine-tuning*. Running fine-tuning for an extended period of time (*extended fine-tuning*) after a single pruning phase can recover significant amounts, if not all, of a network's lost accuracy.

In *standard pruning*, extended fine-tuning to recover lost network accuracy is typically performed only at the end after a single pruning phase [6] as it is time consuming to constantly train pruned networks. In this work, we propose a new *Extended Pruning* procedure in which there are multiple pruning phases and extended fine-tuning is performed at intervals to maintain network accuracy for longer. The motivation for our procedure is that a single long pruning phase may reduce network accuracy more than can be recovered by extended fine-tuning just once at the end [7], whereas by pruning with multiple extended fine-tuning phases to recover lost accuracy we can prune networks to a greater extent than would otherwise be feasible.

Standard pruning has typically been performed on classification networks only and has only been evaluated by us on the potentially more challenging case of an object detection network [7]. Therefore in this work we evaluate our new Extended Pruning procedure on two object detection networks: a network trained for a small custom dataset and a network trained on the more challenging PASCAL VOC dataset [8] which we expect might not have as much redundancy.

The remainder of this paper will cover existing work in the literature in Section II. We outline the proposed approach in Section III. Section IV details our experimental results and Section V presents our conclusions.

II. RELATED WORK

Object detection networks are evaluated with the mean Average Precision (mAP) metric which consists of the mean of Average Precision (AP) of each class in the dataset. AP summarises the shape of the precision/recall curve for a given class. Correct detections only occur if the object label is correct and the predicted bounding box of the detected object overlaps sufficiently with the corresponding ground truth box. Each ground truth box can only be assigned to one detection, with multiple overlapping detections counting as false positives. The overlap metric is the intersection over union (IOU) and it is the ratio of the intersection area to the union area of the two bounding boxes. An IOU of 0.5 is the expected metric for the PASCAL VOC object detection challenge [8] although more challenging datasets like the COCO dataset calculate the mAP at

978-1-5386-6046-1/18/$31.00 ©2018 IEEE

steps of 0.05 between 0.5 and 0.95. In this work we focus on an IOU of 0.5.

Convolutional Neural Networks (CNNs) are considered to be state-of-the-art for many computer vision tasks, including object detection. Regions with CNN features (R-CNN) [9] was one of the first methods to use CNNs for object detection and scored 66% mAP on the VOC 2007 dataset. Fast R-CNN [10] improved the mAP to 66.9% while reducing both inference time and training time (by 146× and 9× respectively). Selective Search [11] was the computational bottleneck for Fast R-CNN. For this reason, Faster R-CNN [12] replaced Selective Search with the Region Proposal Network, and with this change Faster R-CNN increased mAP to 73.2% on VOC 2007 and could run at 5 fps on an Nvidia Tesla K40 GPU.

Many approaches based on R-CNN use pipeline architectures which consist of a region proposal step followed by a classification step and fine-grained localization. Instead of a separate region or object proposal step, alternative approaches have focused on performing object detection in one single CNN. The first network to perform object detection in a single CNN was the YOLO network [13]. Although YOLO only scored 63.4% mAP it runs at 45 fps on an Nvidia Titan X GPU. YOLOv2 [1] improved on YOLO. It achieved 78.6% mAP at 40 fps and a faster version of YOLOv2 scored 76.8% at 67 fps. The Single Shot Multibox (SSD) [2] network is another single CNN that performs object detection and scored 74.3% mAP at 46 fps.

Pruning has been shown to be effective at reducing the size and increasing the speed of deep neural networks. Initial pruning methods include Optimal Brain Damage [14] and Optimal Brain Surgeon [15] which identify the least significant connections through a second-order Taylor expansion approximation of the change in the loss function resulting from the removal of a particular parameter in the network. Although pruning in CNNs can be carried out such that individual weights are pruned [3], this creates sparse matrices in the network which often requires sparse BLAS (Basic Linear Algebra Subprograms) libraries or even specialized hardware for efficient inference [16]. As an alternative to pruning individual weights, pruning entire kernels from a network has been shown as an efficient method of pruning without introducing any sparsity in the network [6], [17], [18]. The general approach is to prune the least important kernels from the network where importance is assessed by techniques such as absolute weight sum [17], mean activation, Taylor expansion [6], and sparse shrink [18].

Among the pruning methods, pruning by absolute weight sum is one of the easiest to implement in practice. It is a data independent method that can rank the kernels for pruning without access to a dataset (although a dataset is required for fine-tuning as described shortly). The absolute weight sum technique was successful in reducing the computational cost for VGG-16 [19] by up to 34% and ResNet-110 [20] by up to 38% with no significant loss in accuracy [17]. We have previously shown that pruning methods based on Taylor expansion did not outperform an approach based on the absolute weight sum [7].

Fine-tuning is a recognised part of every pruning procedure [6], [17], [18] and involves training the network either after each pruning iteration [6] or at the end of pruning [17] to recover lost accuracy. Molchanov et al. [6] fine-tuned for 100 training iterations between pruning iterations. Li et al. [17] performed fine-tuning for 20 epochs on CIFAR-10 after the complete pruning procedure. Molchanov et al [6] used *extended fine-tuning* at the end of the pruning phase to improve the post-pruning accuracy from 83% to 87% on VGG-16 which had an accuracy of 89.3% before pruning. However, they did not perform any further pruning after the extended fine-tuning. This raises the question, therefore, of whether or not further pruning could be performed after extended fine-tuning to obtain a smaller, faster network. Moreover, could additional phases of extended fine-tuning followed by further pruning be performed for further gain?

In the literature, pruning has been evaluated for image classification problems but not for object detection (as far as the authors are aware) except for our own work [7]. There is reason to suspect object detection networks might be less amenable to pruning as more information is carried through the network to the output layer. We have shown that a standard pruning procedure could be effectively applied to an object detection network trained on a small dataset [7] but it is not yet clear whether pruning can be applied with equal success to an object detection network trained on a more challenging dataset such as PASCAL VOC.

III. Proposed Approach

Our approach starts with the tiny-YOLO network, which is a smaller version of the YOLOv2 network and is state-of-the-art among networks of equivalent computational load [1]. The network consists of nine convolutional layers and six pooling layers and is detailed in Table 1.

TABLE I. Tiny-YOLO Network Architecture

Layer Type	Kernel Size / Stride	Kernels	Input Size	FLOPS
Conv	3×3 / 1	16	416×416×3	155 M
Pool	2×2 / 2		416×416×16	
Conv	3×3 / 1	32	208×208×16	401 M
Pool	2×2 / 2		208×208×32	
Conv	3×3 / 1	64	104×104×32	400 M
Pool	2×2 / 2		104×104×64	
Conv	3×3 / 1	128	52×52×64	399 M
Pool	2×2 / 2		52×52×128	
Conv	3×3 / 1	256	26×26×128	399 M
Pool	2×2 / 2		26×26×256	
Conv	3×3 / 1	512	13×13×256	399 M
Pool	2×2 / 1		13×13×512	
Conv	3×3 / 1	1024	13×13×512	1,595 M
Conv	3×3 / 1	1024	13×13×1024	3,190 M
Conv	1×1 / 1	45	13×13×1024	16 M

The number of floating point operations (FLOPS) in a given convolutional layer excluding pooling is given by:

$$FLOPS = 2HW(C_{in}K^2 + 1)C_{out} \quad (1)$$

where H, W, and C_{in} are height, width and number of channels of the input feature map, K is the kernel size (assumed to be symmetric), and C_{out} is the number of output channels.

The number of FLOPS can be reduced by decreasing feature map height and width, decreasing the size of the kernels, or reducing the number of filter kernels. Decreasing the image input size, reduces the number of FLOPS in each layer, but makes smaller objects more difficult to detect. The tiny-YOLO network uses the smallest kernel size that can be used for feature detection (kernel sizes of 1×1 exist in the literature but are not used for feature detection on single feature maps). Each filter kernel in one layer produces one output feature map (channel) in the following layer so reducing the number of kernels reduces the channels in the output. Pruning reduces the FLOPS required by a network by reducing the number of filter kernels it uses.

A. Pruning Proceedure

Pruning can be applied to individual weights within kernels or to entire kernels. Pruning entire kernels does not require any specialized sparse libraries or hardware and for this reason it is the approach we use. In a CNN one filter kernel in one layer produces one output feature map in the following layer. Each kernel contains $C_{in}K^2$ weights where C_{in} is the channels in the input to the layer and K is the kernel size. Pruning one kernel in one layer not only reduces the number of FLOPS in that layer but also reduces the number of FLOPS in the following layer. The number of FLOPS saved by pruning one kernel is given by:

$$FLOPS = 2H^iW^i\left(C_{in}^iK^2 + 1\right) + 2H^{i+1}W^{i+1}(K^2 + 1)C_{out}^{i+1} \quad (2)$$

where H, I, C_{in}, and C_{out} are the height, width, input channels, and output channels for layer i and $i + 1$.

Our Extended Pruning approach, which incorporates multiple pruning and extended fine-tuning phases is as follows:

1. Evaluate the importance of a set of kernels based on some metric.
2. Remove the least significant k kernels.
3. Fine-tune the pruned network for n_{FT} tuning iterations.
4. Evaluate the network after every m iterations of pruning (steps 1-3). If accuracy is p% below the original mAP, run extended fine-tuning for up to n_{EFT} tuning iterations to recover lost accuracy.

Our Extended Pruning method targets a specific layer to perform pruning on but can also be applied across all kernels in the network once normalization is preformed to account for kernels of different sizes [6]. We use the term pruning iteration to refer to steps 1-3 in our procedure.

Multiple methods exist for determining the importance of kernels. The absolute weight sum method was first proposed by [17] and we use a variant of this method to determine which kernels to prune in a network. The motivation behind this technique is that kernels with the smallest weights tend to produce feature maps with the weakest activations relative to the other kernels in the same layer. For each filter kernel, F, the absolute weight sum, s_k, of kernel k is given by:

$$s_k = \sum_{n=1}^{N} \left|F_n^{(k)}\right| \quad (3)$$

where N is the vectorised length of kernel F. Thereafter pruning removes m kernels with the smallest absolute weight sum values. Each kernel removed from a given layer also removes the corresponding weights from the following layer. The absolute weight sum has the advantage of being a data independent technique and this enables ranking the kernels without looking at the data, which saves time in the pruning procedure.

In this work, we always use the *most-FLOPS absolute weight sum method* [7] which we found better than the alternative of selecting the layer with the most kernels [17]. By targeting the layer with the most FLOPS, the computational load of the network can be more effectively reduced as in some cases layers with large numbers of kernels contribute relatively little to the computational load of the network. Using the most-FLOPS absolute weight sum method, the importance of a set of kernels is evaluated by first selecting the layer with most FLOPS and then using the absolute weight sum to identify the least important kernels in the selected layer.

The number of kernels pruned at each iteration determines how quickly we can reduce the computational load of the network. Pruning too aggressively, however, can reduce the accuracy of the network too much to be recoverable [7]. Therefore we prune only one kernel per iteration.

B. Networks

We evaluate pruning on two different networks based on the tiny-YOLO architecture in Table 1: the SPL network, trained on data from the RoboCup Standard Platform League (SPL) robot soccer domain, and the VOC network, trained on the commonly used PASCAL VOC dataset.

1) SPL Network

The SPL network is based on tiny-YOLO and trained on a custom dataset taken from the robot soccer problem domain [21]. The dataset consists of four distinct classes, ball, robot, goal post, and penalty spot. The dataset is less diverse than the PASCAL VOC dataset, on which tiny-YOLO was trained, so before pruning we make some basic modifications to the tiny-YOLO network to reduce the computational load. Specifically we reduce the input image size from 416 to 288 and the number of kernels in the second last and last convolutional layer from 1024 to 512, and 45 to 27 respectively. These modifications reduced the computational load from 7 GFLOPS to 2.56 GFLOPS [7]. The tiny-YOLO network architecture uses the Darknet reference network[1] for the first 13 layers of the network. The Darknet reference network is trained on the ImageNet dataset for classification and we use these pre-trained weights for transfer learning during training.

The SPL network was trained with a batch size of 32, momentum of 0.9, and a decay of 0.0005. During training we

[1] https://pjreddie.com/darknet/imagenet/

used the same standard data augmentation techniques and multi-scale training used in YOLOv2. The network was trained for 250 epochs and scored 64.5% mAP. In previous work [7] we initially performed pruning without extended fine-tuning on this network for a 4.5× reduction in FLOPS and with extended fine-tuning at the end of the pruning process we improved accuracy from 61% to 64.6% mAP. We use this network as the starting point for extended pruning in this work.

2) VOC Network

The VOC network is trained on the PASCAL VOC dataset. The PASCAL VOC dataset [8] was an open competition for object detection from 2007 to 2012, and has since been replaced by the more challenging COCO dataset [22] for competitions. However, the PASCAL VOC dataset is still widely used as a relevant benchmark in the literature. The PASCAL VOC dataset contains 11,530 images consisting of 20 classes. These classes are divided into six animals (bird, cat, cow, dog, horse, sheep), seven vehicles (aeroplane, bicycle, boat, bus, car, motorbike, train), six household objects (bottle, chair, dining table, potted plant, sofa, TV monitor) and one person class. Only the test data from the 2007 challenge is available to download with later versions of the test data hosted at an online server with a limit number of submissions for evaluation. As our approach requires multiple network evaluations, we use the test 2007 dataset for our network evaluations.

The architecture for the pre-trained VOC network[1] is unmodified from tiny-YOLO in Table I. The VOC network scored 52.9% mAP on the PASCAL VOC 2007 dataset.

Experiments and Results

We use the Darknet deep learning framework for all experiments [23]. First we applied extended pruning to the SPL network. We use the following parameterisation of the extended pruning procedure: $k = 1$, $n_{FT} = 50$, $m = 1$, $p = 2.5\%$, and $n_{EFT} = 10{,}000$. All fine-tuning training uses a batch size of 32.

We choose to only remove $k = 1$ kernels per iteration as fine-tuning struggles to recover lost accuracy the more kernels are pruned. We fine-tuned for the same number of iterations, $n_{FT} = 50$, as [6] after each pruning iteration and we evaluate the network after every pruning iteration, $m = 1$, to prevent losing accuracy. Choosing a $p = 2.5\%$ drop in mAP as the threshold to initiate extended fine-tuning is conservative, but it has been shown that the smaller the network gets, the less able the network is to recover lost accuracy due to pruning [6]. In practice this conservative restriction increases the time taken to prune the network but is somewhat offset by the reduction in computational load already achieved by the pruning process.

Fig. 1 shows the performance of the extended pruning procedure on the SPL network. The figure also shows the initial standard pruning conducted as part of prior work [7] and for this reason indicates that no extended fine tuning was performed during the initial pruning phase, after which the network had been reduced by 4.5×. Thereafter, the figure indicates that frequent phases of extended fine tuning were required to recover lost accuracy. Each data point represents the network after 10 iterations of pruning where each pruning iteration generates a new network. We mark a network as extended fine-tuning if extended fine-tuning was required to recover lost accuracy during any of the previous 10 iterations. It is clear from the figure that extended fine-tuning was required more frequently as the network got smaller, being required at least once every 10 iterations after a 10× reduction in FLOPS is achieved. Through the extended pruning procedure, we achieved a total 19× reduction in FLOPS which is significantly better than the 4.5× reduction achieved previously with standard pruning.

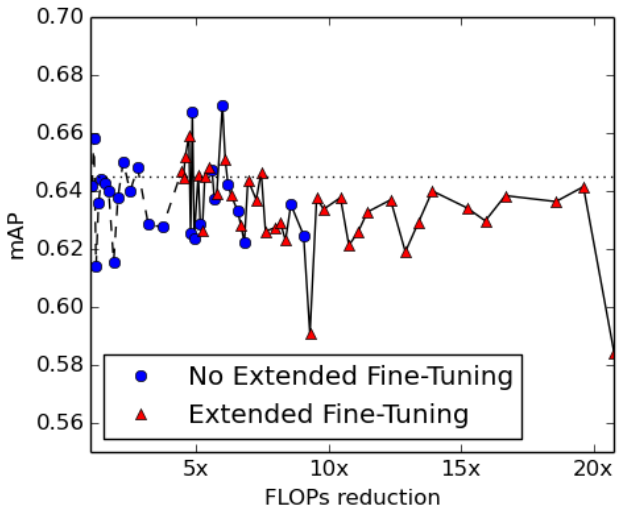

Figure 1: Accuracy of pruned SPL network. The dashed line shows standard pruning, the solid line indicates extended pruning, and the dotted line indicates original network accuracy.

We did not use a strict stopping criterion when pruning the SPL network. A simple stopping criterion for extended pruning could be to stop when extended fine-tuning cannot make the pruned network recover to less than a $p\%$ drop from original mAP. However this simple criterion would have stopped pruning the SPL network at a 9× reduction in FLOPS. As can be seen in Fig. 1, the mAP drops significantly during at this point, but continuing to run extended pruning recovered lost accuracy. A more robust stopping criterion would be if the network failed to recover after multiple extended pruning iterations.

To evaluate how well the extended pruning technique generalizes to more challenging networks and datasets, we applied it to the VOC network. The pruning procedure for the VOC network is broadly similar to the procedure for SPL network, with the following changed parameters: $m = 10$ for the first 500 pruning iterations and $m = 1$ thereafter; $n_{EFT} = 50{,}000$ (due to the larger dataset). During the early pruning iterations we chose to evaluate the network only once every $m = 10$ pruning iterations as we have shown that there is redundancy in the early stages of pruning and evaluating at each pruning iteration is time consuming [7]. After 500 pruning iterations, we evaluate the network after every pruning iteration. We increased the maximum number of iterations for extended fine-tuning to 50,000 to account for the PASCAL VOC being both a larger and more complex dataset.

Fig. 2 shows the effect of extended pruning applied to the VOC network. As in Fig. 1, each data point represents a network after 10 iterations of pruning (where each pruning iterations

[1] Available at https://pjreddie.com/media/files/tiny-yolo-voc.weights

generates a new network) and a network is marked as extended fine-tuning if extended fine-tuning was required during any of the previous 10 pruning iterations. As with the SPL network, extended fine-tuning was required more frequently at latter stages of the pruning process. The VOC network (tiny-YOLO) is already a much smaller network than YOLOv2 and has slightly worse accuracy so it is likely that it initially has much less redundancy than the SPL network relative to its dataset. Nevertheless, the pruning procedure still managed a 1.8× reduction in FLOPS for a 0.8% drop in mAP.

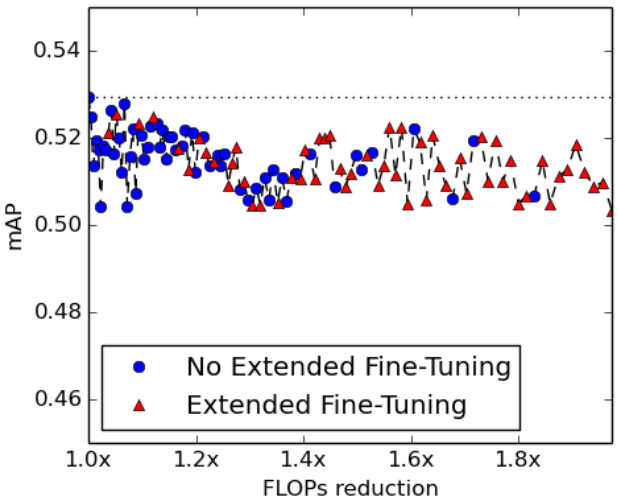

Figure 2: Accuracy of pruned networks on the PASCAL VOC dataset. The dotted line indicates original network accuracy.

Network size is also a significant concern for many applications, as for embedded applications smaller networks can be stored entirely on-chip allowing for fast access and reducing processing latency [3]. In both networks evaluated, the network size was reduced by a greater factor than the FLOPS reduction. The SPL network achieved a 30.5× reduction in network size, while the VOC network achieved nearly a 2.5× reduction in network size.

The results also show that mAP performance does not consistently decrease for the SPL network as kernels are pruned. In some pruning iterations, the pruned networks actually performed better than the initial network. This is not uncommon as the pruning process has been found to help the network generalize to unseen data [24]. Improved network generalization after pruning indicates that the network was originally overfitting the training data [15]. Overfitting can occur when the network is too large for the problem on which it is being trained. In our evaluation we see the network accuracy for the SPL network occasionally outperforms the initial network whereas the VOC network accuracy never exceeds the initial network accuracy. This indicates that the SPL network was overfitting our training data whereas the VOC network was not.

When training on smaller or custom datasets it is common practice to transfer learn from networks trained on the ImageNet dataset as these convolutional features have been shown to generalize well to many image processing tasks [13]. However our results indicate that when working with a smaller dataset, pruning the network might further improve network accuracy by dealing with overfitting problems.

IV. Conclusions

In this work we evaluated the efficacy of our extended pruning method both in terms of reducing the computational load associated with a network trained on a small data set and exploring its effectiveness when applied to a network trained on the more complex PASCAL VOC data set. Our pruning focus was to reduce the computational load of object detection networks while maintaining mAP performance.

We found that the gain associated with pruning in general, and our extended pruning method in particular, is related to the potential redundancy in the network. Both the SPL network and VOC network start with the same architecture. Transfer learning is used to train the SPL network on the relatively small SPL dataset, starting from the modified tiny-YOLO network, but it is likely that this results in a network that has unnecessary computational capacity. Extended pruning can remove the redundancy in that unnecessary capacity. After extended pruning, the SPL network achieved a 94.9% reduction in computational load for a 0.4% drop in mAP. While extended pruning was successfully applied to the VOC network, the 45% reduction in computational load was much more modest and the extended pruning procedure takes longer due to the larger training dataset. Consequently we conclude that pruning in general, and extended pruning in particular, is most applicable to networks that are likely to have substantial redundancy. This makes it particularly applicable when transfer learning is used to adapt an established network architecture designed for a complex dataset to a simpler dataset as can often be the case in embedded system applications.

Acknowledgment

The authors would like to gratefully acknowledge funding provided by the Irish Research Council under their Government of Ireland Postgraduate Scholarship 2013.

References

[1] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 6517–6525, Jul. 2017.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9905 LNCS, pp. 21–37.

[3] G. Manek, J. Lin, V. Chandrasekhar, L. Duan, S. Giduthuri, X. Li, and T. A. Poggio, "Pruning Convolutional Neural Networks for Image Instance Retrieval," *CoRR*, Jul. 2017.

[4] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning Efficient Convolutional Networks through Network Slimming," 2017.

[5] 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)*, 2017, pp. 2755–2763.

[6] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning," in *ICLR*, 2017, no. 2015, pp. 1–17.

[7] S. O'Keeffe and R. Villing, "Evaluating Pruned Object Detection Networks for Real-Time Robot Vision," in *2018 International Conference on Autonomous Robot Systems and Competitions (ICARSC) (Accepted)*, 2018.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[10] R. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[11] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition," *Int. J. Comput. Vis.*, vol. 104, no. 2, pp. 154–171, Sep. 2013.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Nips*, pp. 1–10, 2015.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[14] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems (NIPS)*, 1990.

[15] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks - Conference Proceedings*, 1993, vol. 1993–Janua, pp. 293–299.

[16] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, vol. 16, pp. 243–254.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," in *ICLR*, 2017, no. 2017, pp. 1–10.

[18] X. Li and C. Liu, "Prune the Convolutional Neural Networks with Sparse Shrink," *Electron. Imaging*, pp. 97–101, Aug. 2017.

[19] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *Int. Conf. Learn. Represent.*, pp. 1–14, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[21] S. O'Keeffe and R. Villing, "A Benchmark Data Set and Evaluation of Deep Learning Architectures for Ball Detection in the RoboCup SPL," in *RoboCup International Symposium*, 2017.

[22] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," Springer, Cham, 2014, pp. 740–755.

[23] J. Redmon, "Darknet: Open Source Neural Networks in C." [Online]. Available: http://pjreddie.com/darknet/.

[24] S. Anwar, K. Hwang, and W. Sung, "Structured Pruning of Deep Convolutional Neural Networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, p. 32, Feb. 2017.