18th IEEE International Conference on
Autonomous Robot Systems and Competitions (ICARSC)
April 25-27 2018, Torres Vedras, Portugal

Evaluating Pruned Object Detection Networks for Real-Time Robot Vision

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*—Convolutional Neural Networks are the state of the art for computer vision problems such as classification and detection. Networks like YOLO and SSD have demonstrated excellent results on benchmark datasets such as the PASCAL VOC and COCO datasets. However these networks only run at real time with the support of powerful GPUs and are infeasible for use in low power embedded real-time robotic applications. Pruning has been shown to be an efficient technique for reducing the runtime computational cost of a neural network while maintaining performance in image classification tasks. In this work we evaluate the efficacy of pruning on the problem of object detection using a modified tiny-YOLO network. The network was trained on a custom object detection task and three pruning techniques were evaluated, including our contribution which specifically targets reducing the FLOPS in the network. The results show that pruning with our method followed by extended fine-tuning achieved a 4.5× reduction in FLOPS and a 7× reduction in parameters with no drop in accuracy.**

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

I. Introduction

Multiclass object detection is a key component of applications including autonomous driving, pedestrian detection, and home robots. Unlike image classification, where an entire image is associated with a single label, object detection associates an image with both labels and locations for every detected class within the image. In many applications, such as autonomous driving, object detection needs to be performed accurately in real-time, on platforms that may have low computational power. State-of-the-art object detection methods, however, currently require significant computing power to run in real time.

Convolutional Neural Networks (CNNs) have achieved state-of-the-art results on object detection datasets [1], [2]. These networks can achieve real-time performance on GPUs but are not yet feasible to implement on low-power devices without reducing the computational complexity or increasing the efficiency of the implementation. There are multiple techniques for reducing the computational complexity of a CNN including quantization [3], binarization [4],and depth-wise separable convolutions [5] but this work will focus in particular on network pruning [6].

CNNs have been shown to have a lot of redundancy [7]. Pruning can take advantage of this to reduce the computational cost of the network by removing weights and connections such that the network size and runtime computational load are reduced.

In this work we examine the efficacy of pruning at reducing the size of a network while maintaining object detection performance. Unlike previous work which has primarily evaluated pruning in the context of image classification, we also wish to evaluate if there are any additional considerations when applying this technique to multi-class object detection networks. Although state-of-the-art multi-class object detection networks are trained for a moderately large number of classes (for example, 80 classes for the COCO dataset), there is a demand for specialized applications that handle fewer classes [8]. Our work ultimately targets the Softbank Nao robot, an embedded platform which uses an Intel Atom Z530 CPU and no GPU, and focuses on four object classes drawn from the robot soccer problem domain. These are ball, robot, goal post and penalty spot.

In the remainder of this paper we first outline existing solutions to object detection and pruning in Section II. We outline our proposed approach in section III. Section IV details the experiments performed and the corresponding results. Section V presents our conclusions.

II. Related Work

Performance in object detection is measured with the mean Average Precision (mAP) which is the Average Precision (AP) averaged over all classes in the dataset. AP summarises the shape of the precision/recall curve for detections of a given class where detections are deemed correct only if the predicted bounding box of the detected object overlaps sufficiently with the corresponding ground truth box. The overlap metric is the intersection over union (IOU) and is the ratio of the intersection area to the union area of the two bounding boxes. An IOU of 0.5 is the excepted metric for the PASCAL VOC object detection challenge [9] although more challenging datasets like COCO calculate the mAP at steps of 0.05 between 0.5 and 0.95. In this work we will focus on an IOU of 0.5.

Before CNNs reached state-of-the-art performance for object detection, the best performing approach was Deformable Part Models (DPM) [10]. DPMs are HOG-based graphical models and scored 33% mAP on the VOC 2007 dataset. When CNNs were applied to the object detection problem they exhibited large improvements over DPMs. R-CNN [11] scored

978-1-5386-5346-6/18/$31.00 ©2018 IEEE

66% mAP on the VOC 2007 dataset. Speed and accuracy improvements over R-CNN were made with Fast R-CNN [12], improving mAP to 66.9% while reducing inference time by 146× and with a 9× reduction in training time. Faster R-CNN [13] further improved on Fast R-CNN with the introduction of a Region Proposal Network instead of selective search [14] which was the computational bottleneck for Fast R-CNN. Faster R-CNN increased mAP to 73.2% on VOC 2007 and could run at 5 fps on an Nvidia Tesla K40 GPU.

R-CNN networks in general use a pipeline architecture which consists of a region proposal step followed by a classification step and fine-grained localization. Alternative approaches have focused on performing object detection in one single network with no separate region or object proposal step. The YOLO network [15] was the first network to perform object detection in a single CNN. YOLO was shown to run at 45 fps on an Nvidia Titan X GPU but scored a lower 63.4% mAP than Faster R-CNN. YOLOv2 [1] subsequently improved the accuracy with respect to the runtime. It achieved 78.6% mAP at 40 fps and a faster version of YOLOv2 scored 76.8% at 67 fps. The Single Shot MultiBox Detector (SSD) [2] network takes a similar approach to YOLO achieving 74.3 % mAP at 46 fps.

Pruning has been shown to be effective at removing redundancy in large neural networks. Early pruning efforts included Optimal Brain Damage [16] and Optimal Brain Surgeon [17]. Both these methods identify the least significant connections through a second-order Taylor Expansion approximation of the change in loss function from removing a particular parameter. Pruning in CNNs can be carried out such that either individual weights [18] or entire kernels are pruned. Pruning individual weights creates sparsity in the network which often requires sparse BLAS libraries or even specialized hardware for efficient inference [19]. 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], [20], [21]. The general approach is to prune the least important kernels from the network where importance is assessed by techniques such as absolute weight sum [20], mean activation, Taylor expansion [6], and sparse shrink [21].

Pruning by absolute weight sum is a data independent technique that can rank the kernels for pruning without access to a dataset. This simple technique was successful in reducing the computational costs for VGG-16 [22] by up to 34% and ResNet-110 [23] by up to 38% with no significant loss in accuracy [20]. In contrast, data dependent methods, such as the Taylor expansion method require access to the dataset and can be more complicated to calculate, requiring the gradient of cost function with respect to the activation. The Taylor expansion method reduced computational costs on VGG-16 by 37% [6]. The sparse shrink pruning method tries to find representative data points, such that each point in the dataset can be described as a linear combination of a set of representative data points. The sparse shrink method was applied to Network-in-Network architecture [24], reducing the number of parameters and multiplications by approximately 57% and 74% respectively.

To date it appears that pruning techniques have been evaluated on classification problems. It is therefore an open question if pruning would present any additional issues when applied to object detection networks. In particular, more information is carried through to the output of object detection networks and it might be the case that object localisation is more affected by pruning than classification would be. Indeed some preliminary work we did suggested just such an effect and, for this reason it is important to evaluate if object detection networks can be pruned to the same extent as classification networks. In addition, pruning methods have typically been evaluated on very large networks such as VGG-16 Net. Therefore it is worth examining if pruning techniques can be similarly successful on smaller networks, such as the tiny-YOLO network.

III. Proposed Approach

Our overall approach involves modifying the tiny-YOLO network, training this network via transfer learning on a custom dataset, and finally evaluating a number of pruning techniques on this network. We used the Darknet[1] deep learning framework for our experiments [25] as this is the framework used by the tiny-YOLO network.

A. Dataset

Benchmark datasets for image classification typically contain a very large number of images (for example, 1 million in ImageNet [26]). Current datasets for object detection are much smaller with 11,530 images in PASCAL VOC [9], and 200,000 in COCO [27]. Networks trained for PASCAL VOC are generally pre-trained on ImageNet to learn rich features in the initial convolutional layers. We follow a similar approach in this work with our dataset consisting of 4,140 images containing 12,048 object instances (3973 ball, 3944 robot, 2534 goal post, and 1597 penalty spot) [28]. The dataset was gathered from a range of Nao robot camera image logs in various locations and lighting conditions including our lab, RoboCup 2016 indoor and outdoor, and RoboCup 2017. The dataset is divided into training, validation, and testing datasets with a 70:15:15 split.

B. Network

The initial network examined for this work was the tiny-YOLO network. This is a smaller version of the YOLOv2 network and uses the first 13 layers of the Darknet Reference network[1]. As we are interested in reducing the computational load we first examine the computation associated with the tiny-YOLO network.

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

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

where H, W, C_{in} are height, width and number of channels of the input feature map, K is the kernel width, and C_{out} is the number of output channels. Therefore the unmodified tiny-YOLO network requires of approximately 7 GFLOPS for one forward pass of the network.

To reduce the computational load of the tiny-YOLO network before pruning we modify the network in three distinct ways. The first, already proposed as part of YOLOv2, is to reduce the

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

input image size (from 416 to 288). This effectively halves the number of multiplications in each layer. The second step is to reduce the number of kernels in the second last convolutional layer (from 1024 to 512). This layer had the largest number of FLOPS in the whole network and this reduction combined with the input image size reduction reduced the FLOPS in this layer by a quarter. The third and final step was to reduce the number of dimension priors from 5 to 3. (YOLOv2 uses dimension priors, which are similar to anchor boxes, to specify an initial prior width and height for an object relative to the size of an output grid cell.) Reducing the dimension priors has a relatively small effect, removing 40% of the FLOPS from the final layer of the network, which is already the smallest. However, the change has little effect on the accuracy of the network because dimension priors are based on k-means clustering of the bounding boxes in the dataset and the size and shape of bounding boxes for objects in our dataset is not as diverse as those within the PASCAL VOC dataset. The modified network that results from these three changes can still be trained using transfer learning from the Darknet Reference network as none of the corresponding kernels were changed. The architecture of this modified tiny-YOLO network is detailed in Table I. It requires 2.56 GFLOPS for a forward pass through the network.

TABLE I. Modified Tiny-YOLO Network Architecture

Layer Type	Kernel Size / Stride	Kernels	Input Size	FLOPS
Conv	3×3 / 1	16	288×288×3	74 M
Pool	2×2 / 2		288×288×16	
Conv	3×3 / 1	32	144×144×16	192 M
Pool	2×2 / 2		144×144×32	
Conv	3×3 / 1	64	72×72×32	192 M
Pool	2×2 / 2		72×72×64	
Conv	3×3 / 1	128	36×36×64	191 M
Pool	2×2 / 2		36×36×128	
Conv	3×3 / 1	256	18×18×128	191 M
Pool	2×2 / 2		18×18×256	
Conv	3×3 / 1	512	9×9×256	191 M
Pool	2×2 / 1		9×9×512	
Conv	3×3 / 1	1024	9×9×512	765 M
Conv	3×3 / 1	512	9×9×1024	764 M
Conv	1×1 / 1	27	9×9×512	2 M

The modified tiny-YOLO network was trained with a batch size of 32, momentum of 0.9, and a decay of 0.0005. During training we used the same standard data augmentation techniques and multi-scale training used in YOLOv2 including random crops, rotations, and hue, saturation and exposure shifts. The network was trained for 250 epochs after which it achieves 64.5% mAP on our dataset. This result is in line with the 51% mAP measured in an evaluation of tiny-YOLO on the PASCAL VOC dataset [29].

C. Pruning Techniques

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:

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

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

Modern pruning methods have found that an iterative process of pruning and fine tuning keeps the network resilient to pruning [6], [20]. The general approach is as follows:

1. Evaluate the importance of a set of kernels based on some metric.
2. Remove the least significant *m* kernels.
3. Fine-tune this pruned network for a fixed number of iterations.
4. Repeat steps 1-3 until some criterion is met.

We evaluated two different approaches to pruning our modified tiny-YOLO network. The first approach was proposed by [20] and measures the importance of each kernel in a given layer by its absolute weight sum. The motivation for this approach is that kernels with smallest weights tend to produce feature maps with weakest activations relative to 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 in a given layer and the corresponding weights from the kernels of the following layer.

An important aspect of the general absolute weight sum approach is how to select the layer from which to prune. We evaluated two absolute weight sum methods which we refer to as *most-kernels* and *most-FLOPS*. The most-kernels method always selects the layer with most kernels to prune as [20] found that layers with more kernels are less sensitive to pruning. Preliminary testing, however, indicated that in some cases layers with the most kernels contribute relatively little to the computational load of the network. For this reason we created the new most-FLOPS method. This method always selects the layer with the largest number of FLOPS to prune, calculated using (1), and ignores the number of kernels in the layer.

All absolute weight sum methods are data independent and can therefore rank the kernels without requiring the time consuming process of running the data through the network. Data dependent techniques such as Taylor expansion pruning [6] have also been shown to be effective at pruning networks. The Taylor expansion method approximates change in the loss function that results from removing a specific kernel. The importance of each kernel is determined by:

$$\theta_T\left(z_l^{(k)}\right) = \left|\frac{1}{M}\sum_m \frac{\delta C}{\delta z_{l,m}^{(k)}} z_{l,m}^{(k)}\right| \quad (4)$$

where $z_{l,m}^{(k)}$ is activation at position m in the k^{th} feature map in layer l and M is the length of the vectorised feature map. Every feature map $z_l^{(k)}$ is the output from one kernel, $F^{(k)}$, convolved with the input in the previous layer. For a minibatch with $T > 1$ images, the importance is calculated separately for each image example and averaged over T. This approach requires the accumulation of the product of the activation and the gradient of the cost function with respect to the activation, which is already computed as part of the backpropagation step. However, seeing as this method is a data dependant technique it is more computationally expensive than data independent techniques to rank the kernels.

Rather than operating on one layer at a time, the Taylor expansion method evaluates the importance of every kernel in the network. Since kernels from different layers of the network have different scales, we perform l_2-normalization to rescale across the layers:

$$\hat{\theta}\left(z_l^{(k)}\right) = \frac{\theta\left(z_l^{(k)}\right)}{\sqrt{\sum_j \left(\theta\left(z_l^{(j)}\right)\right)^2}} \quad (5)$$

where the sum is computed over all kernels, j, in the layer l. The authors in [6] also introduce a FLOPS regularization technique which reduces the importance of kernels in layers with large computation requirements. The regularization technique they introduce is as follows:

$$\theta\left(z_l^{(k)}\right) = \theta\left(z_l^{(k)}\right) - \lambda\theta_l^{FLOPS} \quad (6)$$

where λ controls the amount of regularization and θ_l^{FLOPS} is calculated for each layer using (1). The authors in [6] use $\lambda = 10^{-3}$ and we use the same value in this work.

IV. Experiments and Results

First we evaluated the most-kernels absolute weight sum method. This was applied iteratively to the layer with the most remaining kernels and each pruning iteration had the following steps:

1. Determine which layer in the network has the most kernels.
2. Calculate the importance of each kernel using the absolute weight sum from (3).
3. Rank the kernels by importance and prune the lowest ranked m kernels from the network.
4. Fine-tune for 50 iterations with a batch size of 32.

Steps 1-4 are repeated for 300 iterations. The number of kernels, m, was chosen to be 5% of the total number of kernels within the layer. This technique (Abs 5K) was effective at pruning the model size of the network but tended not to affect early layers of the network which contained a small number of kernels but required a large number of FLOPs at runtime.

Therefore we evaluated our new most-FLOPS absolute weight sum method. The procedure followed the same steps as the most-kernels variant except that step 1 selected the layer with the most FLOPS to prune. Once again 5% of the kernels in the selected layer were pruned on each iteration (Abs 5F).

Since the new most-FLOPS method proved to be the more effective than the most-kernels method, we then repeated most-FLOPS but removed just 1 kernel per iteration for 1500 iterations (Abs 1F). This approach was consistent with the final method to be tested, the Taylor Expansion method with normalization and FLOPS regularization, which also pruned just one kernel per iteration (Taylor). The steps for Taylor were identical to those of Abs 1F except that the importance in step 2 was calculated according to (4) instead of absolute weight sum.

Fig.1 shows the effect of pruning using all the methods evaluated. To simplify the figure, each data point represents the best performing network from a block of 10 pruning iterations (where each pruning iteration generates a new network). All methods were reasonably robust to pruning for initially but methods which pruned multiple kernels per pruning iteration tended to be less stable. In particular the mAP of the Abs 5K technique degraded early on in the pruning procedure suggesting that pruning cannot target only deeper layers of the network (that happen to have the most kernels). The Abs 5F method maintained reasonable mAP performance for more pruning iterations while the Abs 1F and Taylor methods which only pruned one kernel per iteration were much more robust to pruning. The best performing network after pruning, Abs 1F, exhibited an mAP degradation of 5.2% while achieving a nearly 4.5× reduction in FLOPS.

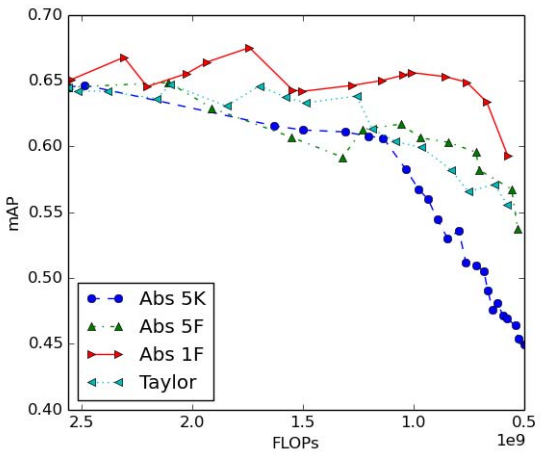

Figure 1: Network accuracy for various methods of pruning. Each data point shows a pruned network.

Perhaps surprisingly, the results showed that our most-FLOPS absolute weight sum method (Abs 1F) outperformed the Taylor series expansion method, achieving better mAP with fewer FLOPS. One possible reason for this is that the Taylor expansion method requires looking at the data, even before fine-tuning to rank the weights. In this evaluation we rank the kernels using just T images per pruning iteration as using the whole dataset on each pruning iteration to get an accurate representation is too computationally expensive. $T = 500$ images was found to be a reasonable number of images for ranking the kernels and is in line with the previous work [6]. In contrast, the absolute weight methods are data independent and the weights themselves become representative of the data that they have been trained on.

It is worth noting that the mAP performance of the evaluated networks does not consistently decrease with decreasing FLOPS. In some pruning iterations, the pruned networks performed even better than the initial network. In general, the increase in mAP is a result of better generalization to unseen test data [30]. Pruning is known to help with overfitting problems [17] which can occur when a trained network has too many weights. Considering that our dataset is smaller than the PASCAL VOC on which tiny-YOLO was originally trained, we expect that the initial network could be overfitting our dataset and this explains why some pruned networks had better accuracy than the initial network.

A. *Pruning effect on Network size*

Pruning to reduce FLOPS also reduces the network size as shown by Fig. 2. For the initial pruning iterations, all techniques show a linear relationship between FLOPS and network size. This is a result of all techniques initially targeting the largest layer, which has both the most FLOPS and the most kernels (and hence parameters), for pruning. However once this layer has been sufficiently pruned, only the Abs 5K technique continues to target layers with of the most kernels (and hence the most parameters) for pruning, even if these kernels are not associated with a large number of FLOPS. The other methods evaluated all target layers with the most FLOPS, whose kernels might not be large (and hence not have so many parameters) but are applied to a larger feature map in earlier layers in the network. The end result is that network size is reduced more slowly by methods which focus on reducing FLOPS.

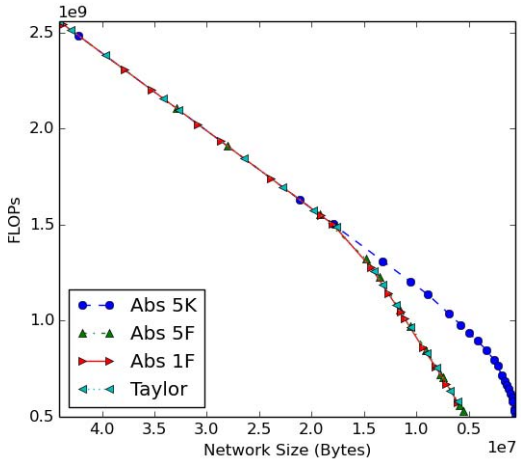

Figure 2: FLOPS of pruned networks for different network sizes. Each data point shows a pruned network.

B. *Extended Fine-tuning*

Pruned networks can be further improved by training for a longer period of time. We trained networks pruned with each of the techniques for a further 75 epochs. Fig. 3 shows the results of this extended fine-tuning. It is clear that fine-tuning can restore some of the mAP performance lost by pruning. However, the more aggressive pruning methods which prune more than 1 kernel per iteration are only able to recover up to a point or, in the case of Abs 5F, not recover at all. Of particular note, extended fine tuning of the pruned Abs 1F network restored the mAP performance of the initial network. This suggests that further pruning could have been attempted with this network, but this has not yet been evaluated.

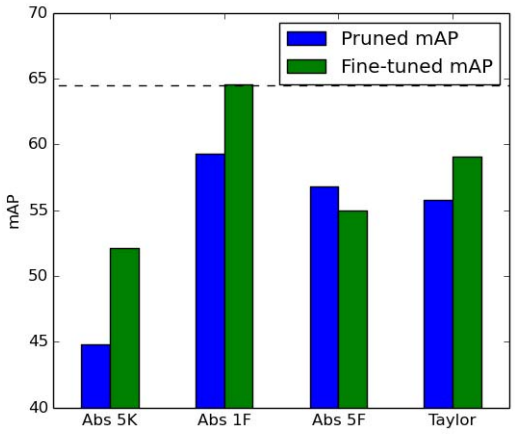

Figure 3: Accuracy improvement of networks under extended fine-tuning. The dashed line shows initial network accuracy

C. *Class Accuracy*

The robot soccer dataset has fewer classes than PASCAL VOC and COCO but has some interesting characteristics and unique challenges. Unlike the PASCAL VOC dataset, where the objects of interest are typically the main object in the dataset images, the objects of interest in the robot soccer dataset can appear anywhere in the image and might be very small in the image.

TABLE II. Class Average Precision for pruned and fine-tuned networks

Network	Ball AP	Robot AP	Goal Post AP	Pen Spot AP
Initial	81.3	80.4	52	44.3
Pruned	81.2	78.6	38	39.3
Fine-Tuned	81.5	80.6	45	51.5

Table II shows a breakdown of the average precision across classes for the initial trained network, the Abs 1F pruned network, and the Abs 1F network after extended fine-tuning. Average Precision for easy classes such as the ball and robot are least effected by pruning. More difficult classes like the goal

post and penalty spot suffer more from pruning but extended fine-tuning can restore lost accuracy.

V. Conclusions

In this work we evaluated the efficacy of different pruning methods to reduce the computational load associated with an object detection network while maintaining mAP performance. We found that pruning, which had been shown to be successful reducing the computational cost of large image classification networks, can be successfully applied to our slightly modified tiny-YOLO network, which is itself a good deal smaller than the state of the art YOLOv2 object detection network. Our new most-FLOPS absolute weight sum method (specifically, Abs 1F) achieved the same mAP as the initial network after extended fine tuning while achieving a 77.5% (almost 4.5×) reduction in computational load. Our results also show that the more complex and time consuming data dependent pruning technique evaluated (Taylor) shows no advantage over our new method. We conclude that this technique may be the most efficient way to prune similar networks in the future. In forthcoming work, we hope to evaluate whether extended fine tuning at the end of the pruning process (as examined here) or periodically throughout the pruning process could permit even greater computational load savings to be achieved. Finally, we note that combining pruning techniques with other methods to reduce computational complexity such as quantization might lead to further improvements in computational load permitting object detection networks to run more easily on embedded processors.

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 *Computer Vision -- ECCV 2016*, 2016, pp. 21–37.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *CoRR*, vol. abs/1609.0, 2016.

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *Eccv*, pp. 1–17, 2016.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.0, 2017.

[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] 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.

[8] K. Lu, X. An, J. Li, and H. He, "Efficient deep network for vision-based object detection in robotic applications," *Neurocomputing*, vol. 245, pp. 31–45, Jul. 2017.

[9] 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.

[10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

[24] M. Lin, Q. Chen, and S. Yan, "Network In Network," *CoRR*, vol. abs/1312.4, 2013.

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

[26] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[27] 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.

[28] 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.

[29] M. Apte, S. Mangat, and P. Sekhar, "YOLO Net on iOS."

[30] 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.