Compressed Object Detection

Gedeon Muhawenayo
African Masters of Machine Intelligence
Kigali, Rwanda
gmuhawenayo@aimsammi.org

Georgia Gkioxari
Facebook AI Research (FAIR)
Menlo Park, CA, US
georgia.gkioxari@gmail.com

Abstract

Deep learning approaches have achieved unprecedented performance in visual recognition tasks such as object detection and pose estimation. However, state-of-the-art models have millions of parameters represented as floats which make them computational expensive and constrain their deployment on hardware such as mobile phones and IoT nodes. Most commonly, activations of deep neural networks tend to be sparse thus proving that models are over parametrized with redundant neurons. Model compression techniques, such as pruning and quantization, have recently shown promising results by improving model complexity with little loss in performance. In this work, we extended pruning, a compression technique which discards unnecessary model connections, and weight sharing techniques for the task of object detection. With our approach we are able to compress a state-of-the-art object detection model by 30.0% without a loss in performance. We also show that our compressed model can be easily initialized with existing pre-trained weights, and thus is able to fully utilize published state-of-the-art model zoos.

1 Introduction

Deep neural networks are computationally expensive. Their memory and compute complexity limit their deployment on edge devices and make them unsuitable for applications with strict latency requirements [19]. The progress in VR, AR, IoT and smart wearable devices create opportunities for researchers to tackle the challenges of deploying deep neural networks on devices with constrained memory, CPU, bandwidth and energy [11, 14].

To this end, compression techniques have shown promising results in reducing the memory and time requirements of deep neural networks. Pruning reduces the model's complexity by removing unnecessary elements in their structures at different levels and without a significant drop in accuracy [17]. Quantization converts a model to use reduced precision integer representation for the weights or activations. Quantizing a model helps reduce its size leading to higher throughput of operations on CPU or GPU, and improved inference speed. Operationally, quantization is achieved by multiplying the floating point parameters with a scale factor and rounding the output to its nearest integer. Quantization approaches differ in the way the scaling factor is determined [15, 10].

In this work, we apply pruning and quantization techniques for visual recognition tasks. We apply these compression techniques on the convolutional neural networks which commonly comprise the backbone architectures of state-of-the-art recognition models such as in Faster RCNN [22]. There, 60% of the model parameters belong to the CNN backbone. Model compression on CNNs is well suited as recent studies [17] have shown that CNN structures have redundant parameters which can be removed without loss in performance. In addition to pruning, we further improve the efficiency of object recognition models by quanitizing their parameters from 32-bit float to 8-bit integer precision. Finally, we show that our compression techniques also support pre-traiend model weight initialization thus making it possible to take advantage of published model zoos.

2 Related work

Object Detection. Object detection has witnessed tremendous progress in recent years due to its wide range of applications and recent technological advancement [11]. The task has found wide practical applications such as self driving cars, robotic vision, user content understanding and much more [1]. The success of object detection is mainly attributed to recent breakthroughs in deep neural networks and hardware, namely Graphical Processing Units (GPUs) [11]. Recent state-of-the-art object detectors [4, 22, 8, 6] break the problem into two stages. First, a CNN proposes regions of interest on the input image. Second, another network extracts features from the input image and the detected regions and performs object classification and localization or pose estimation. Faster R-CNN, an object detection approach proposed in [22], follows the aforementioned two-stage recipe and is comprised of a CNN backbone followed by a region proposal network (RPN) and a region classifier.

Undoubtedly, the CNN backbone is an essential part of object recognition systems as it embeds the input image into a higher dimensional feature space. Recent breakthroughs in CNN architectures has pushed performance to new highs. Feature Pyramid Networks (FPN) proposed in [13] draw lateral connections between different layers of the CNN and have now became an important building block of state-of-the-art object detection systems. The lateral connections in FPNs fuel all scales of a CNN with semantics from the downstream task and enable detection of objects in a variety of scales.

Detectron2. In 2019 Facebook AI Research open sourced the next generation object detection platform known as Detectron2 [23]. It can localize, recognize and predict attributes for every object in an image providing a complete understanding of every pixel in the image. Detectron2 is built in Pytorch, it is a ground up rewrite of Detectron [5] with faster speed, more accurate models and more modular design. Detectron2 includes Faster R-CNN, Mask R-CNN, RetinaNet and other popular recognition models in the computer vision community.

Detectron2 features a wide variety of object recognition tasks including object detection where each object is detected with a label and a bounding box, instance segmentation where each object is marked with its 2D silhouette, human pose estimation where for each detected human its landmarks (e.g. right shoulder) are localized and panoptic segmentation which predicts things and stuff (e.g. sky). Along with these tasks, Detectron2 provides a comprehensive model zoo of pre-trained weights for each tasks and for different backbone architectures (e.g. ResNet50, ResNet50FPN etc.)

Detectron2 is flexible to use and it is designed in way that enables the user to import it as a normal library and build preferred customization on the top of it. Using the registration provided user can replace its backbone, add new type of head, use a custom dataset, or customize other components in the system. Nonetheless, most models in Detectron2 are heavy with millions of parameters and thus hard to deploy on resource-constrained devices.

Model Compression. Recent techniques of compacting and accelerating deep neural network models are classified into four broad categories: parameter pruning and quantization, low-rank factorization, transferable/compact convolutional filters and knowledge distillation [2].

- **Network pruning.** The main goal of network pruning is to discard redundant, non-informative weights in a pre-trained DNN model. The deep compression method in [2] removes the redundant connections and quantizes the weights. Huffman coding is then used to encode the quantized weights. In recent works, there is a growing interest in sparsifying model weights during training. Those sparsity constraints are typically introduced in the optimization problem as L_n-norm regularizers. An early approach to pruning was the biased weight decay [2]. Magnitude based pruning [16] is among the most widely used approaches in pruning. The Optimal Brain Damage and the Optimal Brain Surgeon methods [7] that reduce the number of connections based on the Hessian of the loss function has been proven to give sparse model with good accuracy [2]. The Soft Filter Pruning (SFP) method proposed in [9] accelerate the inference procedure of deep Convolutional Neural Networks (CNNs) by enabling the pruned filters to be updated when training the model after pruning [2].
- **Quantization and Binarization.** Model quantization compresses the original network by reducing the number of bits required to represent each weight [3]. [2] showes that 8-bit quantization of the model weights can result in significant speed-up with minimal loss of accuracy. For weight initilization, quantization can be applied to the pre-trained

weights or the weights can be quantized during training. [3] only quantizes a different random subset of weights during each forward pass, allowing for unbiased gradients to flow through the weights. Binarization, an extreme case of quantization also referred to as binary neural networks, represents each weight by a single bit. Assuming a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$ in a binary neural network, each entry $W_{ij} \in \{0, 1\}$. Binarizing the model largely saves storage and compute, and it is a promising technique for deploying deep models on resource-constrained devices. However it results in very poor performance compared to other compression techniques.

It has been suggested that pruning, quantization and low-rank factorization are the best compression techniques to use when working with pre-trained models [2]. Evidence so far suggests that parameter pruning and quantization give good results with little loss in performance.

3 Methods and Experiments

One of the main facts that make deep models so compressible is their redundancy in parameters [17]. Specifically, all of the CNN structures have redundant weights which can be removed. The redundancy is originated from the brain mechanism, which can recover its functional capability even in the existence of reasonable neural damage [17, 12]. Our approach compresses the model parameters by introducing sparsity into the weights and using few bits to represent each parameter with marginal performance sacrifice.

3.1 Pruning

Pruning generally removes the parts of the model that contribute less or nothing to the final performance. The result is a sparse model that can be stored efficiently and with an high inference speed with minimal loss in accuracy.

Weight pruning. in this approach, to achieve a sparsity percentage of k% (i.e. remove k% of the connections) we rank the individual parameters in the parameter matrix w according to their magnitude (absolute value), and then set to zero the smallest k% of the weights. We use L_n norm to measure their magnitude. Setting an individual parameter to zero is equivalent to deleting connections.

Unit/Neuron pruning. In this approach, we set entire columns in the parameter matrix to zero. This corresponds to completely removing a neuron. We use L_2-norm to rank the columns of the parameters matrix. L_2-norm helps to achieve a sparsity of k% (i.e. remove k% of the nodes), finally we delete the smallest k% columns.

Pruning can be cast as an optimization problem

$$\min_w L(w; D) = \min_w \frac{1}{n} \sum_{i=1}^{n} l(w; x_i y_i) \quad \text{s.t.} \quad \|w\|_0 \leq k \tag{1}$$

where w are our model parameters, $D = \{x_i, y_i\}_{i=1}^n$ is our dataset, $l(w; x, y)$ is our loss function and k represents the desired sparsity level in the parameters.

Pruning converts a dense model into a sparse model, it only keeps important connections to induce sparsity into the model weights. Pruning is effective in reducing the network complexity and addressing the over-fitting problem.

Generally, a compressible weight vector $\in R^n$ may be written as a sparse vector $s \in R^n$ (containing mostly zeros) in a transform basis $\Psi \in R^{n \times n}$.

$$w = \Psi s \tag{2}$$

Actually, when high-dimensional signals exhibit low-dimensional structure, they admit a sparse representation in an appropriate basis or dictionary. In addition to a signal being sparse in an SVD or Fourier basis, it may also be sparse in an overcomplete dictionary whose columns consist of the training data itself.

Neurons and Synapses importance. Neurons/synapses are removed based on the magnitude of their weights [16]. Mostly, parameters with low magnitude are removed. The other technique is to prune neurons/synapses based on activations, gradients or custom rules for neural importance.

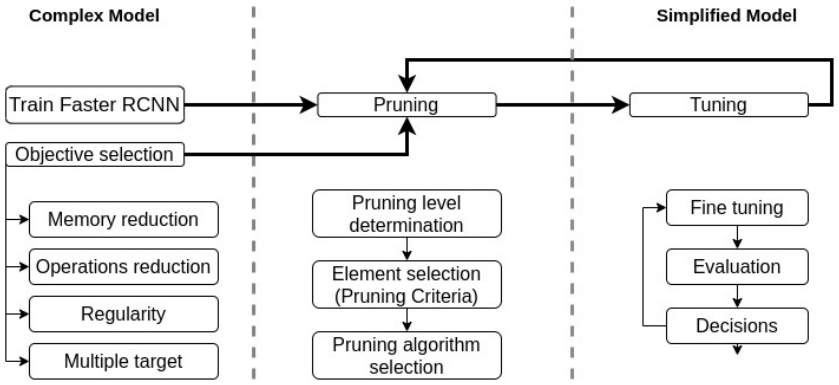

Figure 1: Pruning framework.

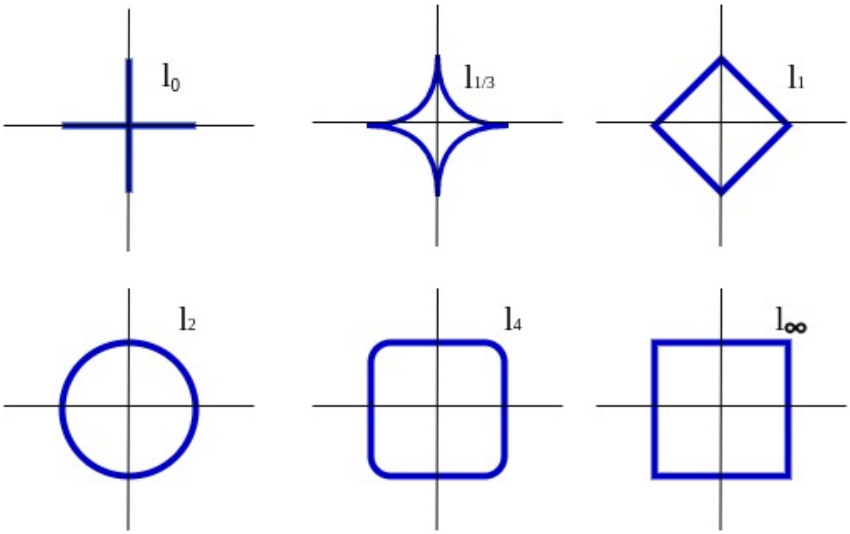

Figure 2: The geometric properties of various norms

Local vs. global pruning. Local pruning consists of removing a fixed percentage of units/connections from each layer by comparing each unit/connection exclusively to the other units/connections in the layer. On the contrary, global pruning pools all parameters together across layers and selects a global fraction of them to prune. The latter is particularly beneficial in the presence of layers with unequal parameter distribution, by redistributing the pruning load more equitably. A middle-ground approach is to pool together only parameters belonging to layers of the same kind, to avoid mixing, say, convolutional and fully-connected layers.

Unstructured vs. structured pruning. Unstructured pruning removes individual connections, while structured pruning removes entire units or channels. Note that structured pruning along the input axis is conceptually similar to input feature importance selection. Similarly, structured pruning along the output axis is analogous to output suppression

The use of the l_1 norm to promote sparsity significantly predates model compression. In fact, many benefits of the l_1 norm were well-known and oft-used in statistics decades earlier.

3.2 Experimental setup

Before applying any compression techniques, using our custom dataset we consider the training of a faster RCNN with a Resnet50 backbone and with FPN. Starting with the pre-trained weights of the Detectron2 detection model zoo trained on the COCO dataset and which has inference speed of 38ms/im, average precision of 40.2 and it requires 3GBs memory to train.

Dataset. We have collected our dataset from East African parks, it contains 1309 instances. The following dictionary describes the categories of animals that we are aware of and their number of instances into the dataset. Keys represent the animal category while the value represent the number of instances per that category **{'giraffe': 101, 'person': 152, 'zebra': 131, 'elephant': 166, 'impala': 169, 'monkey': 80, 'lion': 108, 'leopard': 63, 'crocodile': 61, 'buffalo': 97, 'hyena': 70, 'bird': 123, 'gorilla': 88}**. We split our dataset into training and validation on a ration or 80% and 20% respectively.

Model. Following the common experimental setting in related work on network pruning in [17], we extended their approaches from image classification to the object detection model with Faster RCNN architecture. Our approaches can be used on other object detection models such as YOLO [21] [20] among others. Our model has three main blocks; backbone, proposals generator and ROI heads. The whole model has a total of 41.4 Millions of trainable parameters.

	Model Size			
	Backbone	Proposals generator	ROI heads	Total
Trainable parameters	26.8M	0.6M	14.0M	41.4M
Size on Memory	107.2MBs	2.4MBs	56MBs	165.6MBs

Table 1: Parameters per Faster RCNN block.

The backbone has more than 60% of the total parameters, this makes it the most targeted block in our compression experiments.

We take advantage of the publicly available state-of-the-art object detection models in Detectron2 and its model zoon. We use the Pytorch pruning library [18] which works really well when the model is fully implemented in Pytorch. To improve the performance of our approach with marginal accuracy drop, we treated each block with in our model as a compression task, so global pruning here refers to the global parameters of a particular block.

To sparsify our weight tensors we had first to determine the weights that do not contribute much on the model performance. Early work showed that parameters magnitude could be a good measure to know how each individual parameter contributes to the final activation. Generally but not always, parameters with small magnitude do not contribute too much to the model performance, this makes L_n-norm one of the good techniques to sparsify the model parameter tensors.

Pruning Mask. After comparing the weights in our model parameters, a pruning mask of the same shape as our parameters and whose values $\in \{0, 1\}$ is generated, where 0 represent parameters to be removed and 1 represent the parameters to keep.

L1 Global Unstructured: The magnitude of parameters is measured using L1-norm and the comparison is between individual parameters. Depending on the percentage of parameters to be pruned, individual connections starting from the one with smallest magnitude are removed. The pruning mask is updated with 0's representing pruned parameters and 1's otherwise. Modular wise the global pruning cares only on the percentage to be pruned and the parameters with small magnitude, this means more parameters might be pruned from a single module depending on how they would fire activation. This makes global pruning a good approach as it globally remove unnecessary parameters.

L_n norm Global structured/unstructured: When it is structured, the entire units or channels would be removed depending on their magnitude measured by using L_n norm, and the percentage of parameters we want to prune. The parameters of small magnitude will be pruned first. In case of Unstructured the only difference is how we compare, we remove individual connections instead of removing the channels or units.

Random structured/Unstructured: With this method we randomly prune a percentage of parameters. This method might be structured or unstructured. The main problem with this approach, parameters/connections are pruned randomly, there is no any measure on parameters importance.

3.3 Experiment Results

Regardless of the goal, pruning imposes a trade-off between model efficiency and quality, with pruning increasing the former while (typically) decreasing the latter.

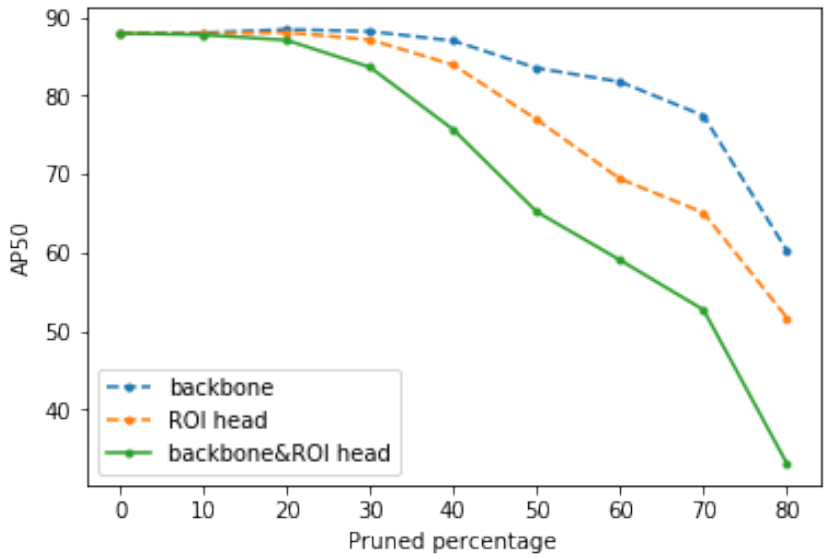

Figure 3: AP50 for our compression.

Table 2 shows a comparison of different methods of pruning used, For low sparsity our approaches outperforms even the dense baseline, which is in line with regularization properties of network pruning. On large models, pruning shows reasonable performance even with extremely high sparsity level.

	Pruned percentage in the backbone								
	0%	10%	20%	30%	40%	50%	70%	80%	90%
AP50	87.92	87.97	88.40	88.15	86.95	83.48	77.42	60.24	0.17
Memory(MBs)	165.6	154.88	144.16	133.44	122.72	112.0	90.56	79.84	69.12
	Pruned percentage in the ROI head								
	0%	10%	20%	30%	40%	50%	70%	80%	90%
AP50	87.92	87.97	88.40	88.15	86.95	83.48	77.42	60.24	0.17
Memory(MBs)	165.6	160.0	154.4	148.8	143.2	137.6	126.4	120.8	115.2
	Pruned percentage in the both backbone and the ROI head								
	0%	10%	20%	30%	40%	50%	70%	80%	90%
AP50	87.92	87.72	87.04	83.60	73.64	61.18	54.69	33.12	0.14
Memory(MBs)	165.6	149.2	132.9	116.6	100.3	84.0	51.3	35.0	18.7

Table 2: AP50, Model size results of the used Pruning approach (L_1 Unstructured) and quantization.

4 Conclusion

We show that pruning and quantization techniques can efficiently compress object recognition models with little loss in performance. We can prune 40% of the model with loss of a few points in average precision. The reduction in memory allows for efficient storage and enables deployment of object detectors on devices of lower computational capacity.

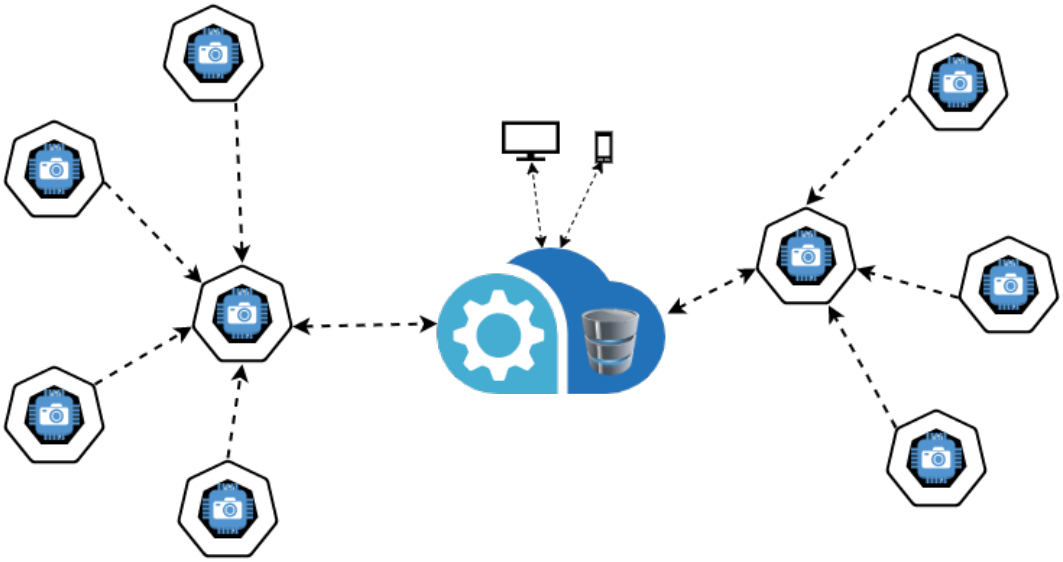

Figure 4: Distributed AI powered IoT nodes.

Broader Impact

Africa's tourism industry is now the second fastest growing in the world. Some 67 million tourists visited Africa in 2018, representing a rise of 7% from a year earlier. Tourism is an important economic sector for many countries in Africa. The touristic particularity of Africa lies in the wide variety of points of interest, mainly Parks, and multitudes of landscapes as well as the rich cultural heritage. Visiting the park does not guarantee to see animals that you wanted to see because sometimes even the guards they don't really know where actually the animals are located because most of the parks cover a huge geographical area. They try to use cars to travel into the park so that they could see animals that they want to visit, however most of the time they do not really know where animals are located.

With the animals detection model and compression techniques explored in our work as well as the dataset used, our work can contribute to the modernization of tourism in Africa and worldwide. As the compressed model is light, with little modification it can be deployed on distributed IoT nodes, which constantly update a dashboard indicating where a particular types of animal is located. Those nodes should be powered by solar energy and they shouldn't rely on the internet connectivity, furthermore to make the system affordable, they should be implemented with low cost sensors and components like Raspberry pi etc.

How it could work: Assume that we have deployed an animal detection model in each node, and the model is able to detect different types of animals found in that park and as it detects an animal it should update the dashboard saying that at this time, this type of animal is located at this location.

References

[1] Steven L. Brunton and J. Nathan Kutz. *Dimensionality Reduction and Transforms*, page 1–2. Cambridge University Press, 2019.

[2] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks, 2017.

[3] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression, 2020.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[5] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[6] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn, 2019.

[7] Babak Hassibi, David G. Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 263–270. Morgan-Kaufmann, 1994.

[8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017.

[9] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks, 2018.

[10] Sambhav R. Jain, Albert Gural, Michael Wu, and Chris H. Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks, 2019.

[11] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.

[12] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[13] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[14] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning, 2018.

[15] Prateeth Nayak, David Zhang, and Sek Chai. Bit efficient quantization for deep neural networks, 2019.

[16] Michela Paganini and Jessica Forde. On iterative neural network pruning, reinitialization, and the similarity of masks, 2020.

[17] Morteza Mousa Pasandi, Mohsen Hajabdollahi, Nader Karimi, and Shadrokh Samavi. Modeling of pruning techniques for deep neural networks simplification, 2020.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, Sep 2020.

[20] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[21] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[23] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.