Compressed Objects Detection

Gedeon Muhawenayo
African Masters of Machine Intelligence
Accra, Ghana
gmuhawenayo@aimsammi.org

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

Abstract

Deep learning approaches have achieved unprecedented performance in recognition tasks such as object detection and facial recognition among others. However, those models have millions of floating parameters, which make them computational expensive and constrain their deployment on hardware such as mobile phones and IoT nodes. There has been a lot of work on model compression techniques such as pruning, quantization, distillation and binarization. It has been proved that in deep models there will always be redundant parameters and neurons. This makes pruning one of the promising compression techniques as it only keeps important connections within a model. In this work we extended pruning and weight sharing techniques to the faster RCNN with resnet50 backbone. With our approach we were able to compress the objects detection model by 30.0% without losing the performance. This approach could be a goodway to go when you are using pretrained models.

1 Introduction

Deep neural networks are computation expensive and memory intensive, these limit their deployment on edge devices and applications with strict latency requirements. The progress in VR, AR, IoT and smart wearable devices, create opportunities for researchers to tackle the challenges of deploying Deep Neural Networks models on edge devices, with constrained memory, CPU, bandwidth and energy.

Recently by pruning techniques, significant parts of the computational operations of models could be reduced. Pruning methods reduce model complexities by removing unnecessary elements in their structures. Pruning can be used in different levels of model to decrease the utilized parameters with no significant accuracy drop. Quantizing an neural network means converting it to use a reduced precision integer representation for the weights and/or activations. Quantizing a model helps to reduce the model size and it allows the use of higher throughput math operations on CPU or GPU, this improves inference speed. On a high level, quantization is a conversion from floating point to integers values, it is done by multiplying the floating point parameters by a scale factor and finally the results are rounded to a whole number. Quantization approaches differ in a way scaling factor is determined.

Convolutional Neural Network based backbone is one of the building blocks in the Faster RCNN object detection framework. Looking at the total number of parameters in the Faster RCNN models 60% of them are from the CNN backbone. Recent studies on parameters redundancy in the CNNs have shown that All of the CNN structures have redundant parameters which can be removed. Pruning could be modeled as an optimization problem $\begin{array}{ll} \min_w L(w;D) = \min_w \frac{1}{n}\sum_{i=1}^{1}(w;(x_i y_i)) \\ s \cdot t \cdot w \in R^m \quad \|w\|_0 \leq k \end{array}$

Where $D = f(x;y)$ is our dataset, i rang from 1 to n and k represent the sparsity level in the parameters.

Our contribution was to extend the pruning framework proposed in[] to faster RCNN object detection models, and to improve the pruned version by quantizing the model parameters from float32 to int8 representation.

2 Related work

Objects Detection: Under both the academia and industries, the task of objects detection has been pulling in expanding measures of consideration in recent years due to its wide range of applications and recent technological advancement. This recognition task has found wide practical applications such as surveillance systems, drone scene analysis, Self driving cars, and vision of robots. The success of Object detection was mainly attributed by recent breakthroughs in deep neural networks and and Graphical Processing Units (GPUs).

Recent state-of-the-art object detectors utilize two stages, the first is a CNN based backbone to propose regions of interest and the second is a detection network to extricate features from input images, Classification and Localization.

Faster R-CNN, an object detection framework proposed by S. Ren et al, shortly after the Fast RCNN. It is made up of two modules. The first module known as backbone is a deep CNN that proposes regions of interest, and the second module which is the Fast R-CNN detector uses the proposed regions to extract features, classify and localize objects. From Fast RCNN to Faster RCNN most individual blocks of an object detection system are integrated into a unified end-to-end system. In this framework the RPN module tells the Fast R-CNN module where to look by using the attention mechanism.

The Feature Pyramid Networks (FPN) proposed by T.-Y. Lin et al, has now became a basic building block of the most object detection systems. Actually, features in deeper layers of a CNN are gainful for recognizing categories. However, Before FPN most of the proposed objects detection frameworks run detection only on a network's top layer, because pyramids are computational expensive and memory intensive. To construct feature pyramids with small extra cost, a topdown architecture with lateral connections is developed for building high-level semantic feature maps at all scales. Since a CNN naturally forms a feature pyramid through its forward propagation, the FPN shows great advances for detecting objects with a wide variety of scales.

Model Compression: Recent techniques of compacting and accelerating deep neural network models are classified into four broad categories; Parameters pruning and quantization, Low-rank factorization, Transferable/compact convolutional filters and Knowledge distillation.

- **Network pruning**: The main goal is to prune redundant, non-informative weights in a pre-trained DNN model. The deep compression method in [] removed the redundant connections and quantized the weights, and then used Huffman coding to encode the quantized weights. In the 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. The early approach to pruning was the Biased Weight Decay []. Magnitude based pruning is among the most widely used approach. However, The Optimal Brain Damage [] and the Optimal Brain Surgeon [] methods that reduce the number of connections based on the Hessian of the loss function has been proven to give sparse model with good accuracy. The Soft Filter Pruning (SFP) method proposed in [] accelerate the inference procedure of deep Convolutional Neural Networks (CNNs) by enabling the pruned filters to be updated when training the model after pruning.
- **Quantization and Binarization**: Model quantization compresses the original network by reducing the number of bits required to represent each weight. Vanhoucke et al. [] showed that 8-bit quantization of the model weights can result in significant speed-up with minimal loss of accuracy. Qantization can be applied to pretrained models or the weights can be quantized during training. Angela Fan et al only quantized a different random subset of weights during each forward, allowing for unbiased gradients to flow through the other weights.

  Binarization also referred to as binary neural network, represent each weight by a single bit. Assuming a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$ in binary neural network each entry $W_{ij} \in \{0, 1\}$.

Binarizing the model largely save the storage and computation, and it is a promising technique for deploying deep models on resource-constrained devices, however it results in very poor performance as 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 pretrained models. Early works showed that parameters pruning and quantization generally gives reasonable results while not hurting the model performance.

Detectron2: In 2019 Facebook AI Research open sourced the next generation object detection platform known as Detectron2. It can localize, recognize and predict attributes for every object in an image to give a complete understanding of every pixel in the image. Before Detectron2 objects detection was about predicting bounding boxes and category labels. Detectron2 is completely build in Pytorch, it is a ground up rewrite of Detectron with faster speed, more accurate models and more modular design. It can predicts key-points of human and their poses and label their body pixels, this is known as dense pose prediction. It can segment objects in the scene , label pixels in the background.

Dectron2 has complicated object detection models, which have operators that works on tensors of dynamic shapes. The results of Detectron2 can reproduced by accessing the pretrained models in the public model zoo. In Detectron2, input image goes through a CNN backbone to extract some features that are used to predict regional proposals(regions that are likely to contain objects), features in those objects are cropped and wrapped into regional features then different types of prediction head use regional features and image features to predict boundboxes, segmentation mask for each instance, keypoints and dense pose for each human found in the image. It can also predict semantic segmentation of the entire image and some predictions can be combined to predict panoptic segmentation where it segment all instances and background pixels.

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 custom dataset, or customize other components in the system.

3 Methods and Experiments

One of the main facts that make deep models so compressible is their redundancy in parameters[]. 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 []. Our architecture compress 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 performance. It results in 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 technique, we set entire columns in the parameter matrix to zero, This correspond to the deleting the output of 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.

We modeled pruning as an optimization problem $\begin{array}{ll} \min_w L(w; D) = \min_w \frac{1}{n} \sum_{i=1}^{1} (w; (x_i y_i)) \\ s \cdot t \cdot w \in R^m \quad \|w\|_0 \leq k \end{array}$

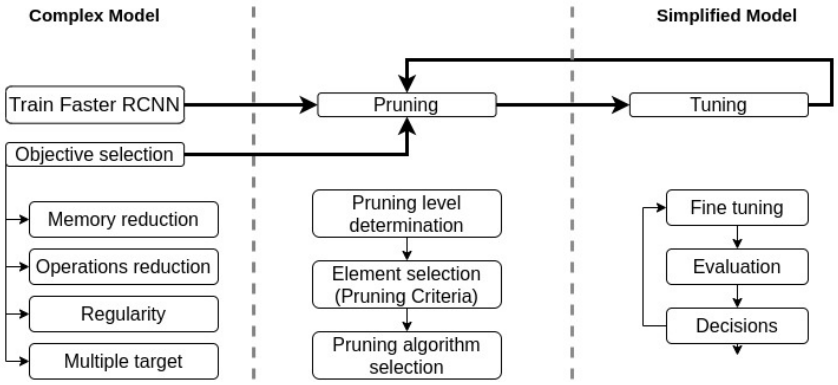

Figure 1: Pruning framework.

Where $D = f(x; y)$ is our dataset, i rang from 1 to n and k represent the sparsity level in the parameters.

Pruning Change 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 $\mathbf{w} \in R^n$ may be written as a sparse vector $\mathbf{s} \in R^n$ (containing mostly zeros) in a transform basis $\Psi \in R^{n \times n}$.

$$\mathbf{w} = \Psi \mathbf{s}$$

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. 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.

Local vs. global: 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: 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 whose a Resnet50 backbone and with FPN. Starting with the pretrained weights of the Detectron2 detection model trained on 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

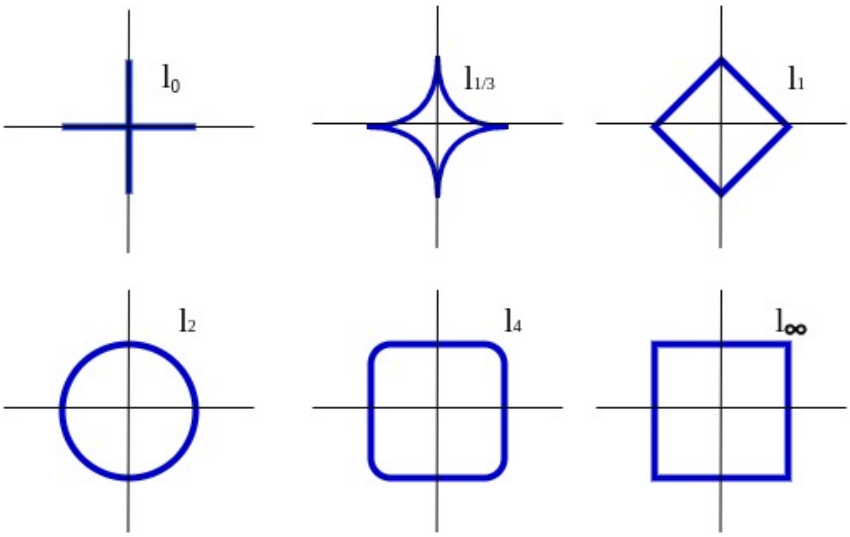

Figure 2: The geometric properties of various norms

instances into the dataset. Key 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 [], we extended their approaches from image classification to the objects detection model with Faster RCNN architecture. Our approaches can be used on other objects detection models such as YOLO[] 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.

Taking advantage of public available state of the art objects detection models in the Detecron2 model zoo. We did our implementation on the top of Detectron2, this gave us advantage of starting with their weights and using the Pytorch pruning library 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, then global pruning here refers to the global parameters of a particular block.

To sparsify our parameter tensors we had first to determine the parameters that do not contribute to much on the model performance. Early work showed that Parameters magnitude could be a good measure to know how each individual parameters fires the 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.
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).

4 Conclusion

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

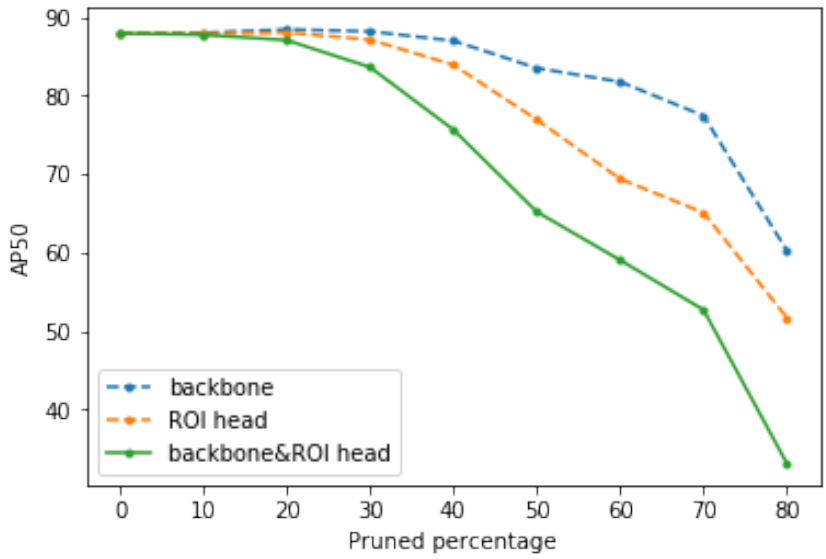

Figure 3: AP50 for our compression.

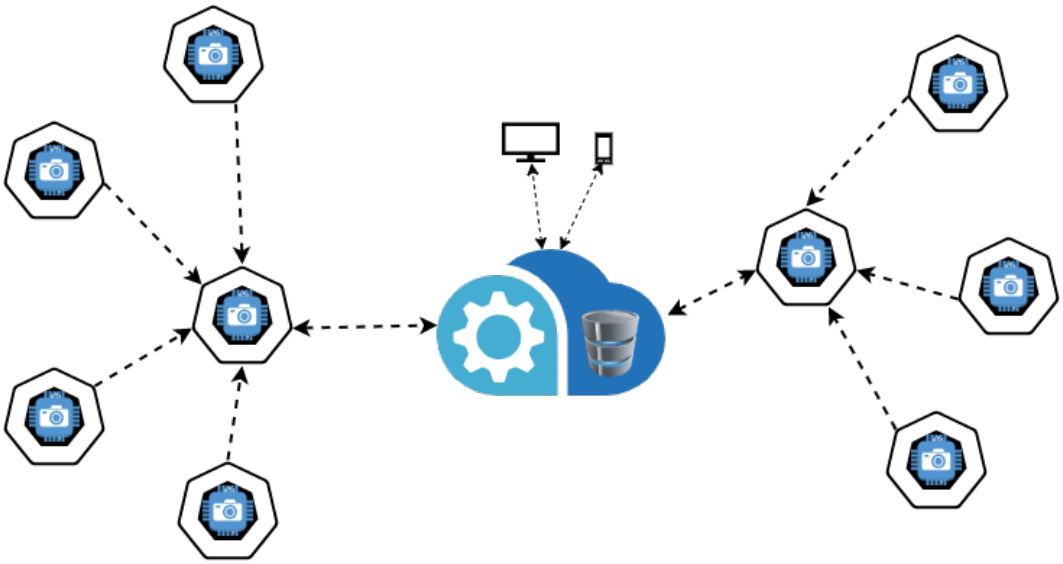

Figure 4: Distributed AI powered IoT nodes.

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