Model Compression in Object Detection

Andrey de Aguiar Salvi* and Rodrigo C. Barros†
Machine Learning Theory and Applications Lab (MALTA)
School of Technology, Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, RS, Brazil
*andrey.salvi@edu.pucrs.br, †rodrigo.barros@pucrs.br

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533792

***Abstract*—Compressed neural-network models have a growing relevance in the Deep Learning literature, since they allow the deployment of AI on devices with computational constraints for many automation purposes. Despite the amount and diversity of work for this purpose, there is no standard benchmark in the literature, and it is often difficult to choose the proper approach due to the large difference between models, datasets, and training details that are tested. Therefore, this paper proposes a standard experimental benchmark for different model compression approaches for the object detection task, using a fixed model (the well-known YOLOv3) and training scheme. Between Pruning, Knowledge Distillation, and Neural Architecture Search, our experiments reveal that the best trade-off is by using pruning, which enables the creation of a model with 80.67% mAP of the original model but removing 98.8% of the parameters, 96.53% of the Multiply–Accumulate Operations, and reducing the storage size from 235.44MB to 11.61MB.**

***Index Terms*—Pruning, Knowledge Distillation, Neural Architecture Search, Model Compression, Object Detection**

I. Introduction

Object detection is the task of recognizing objects and retrieving their image localization, being useful for automation on scenarios requiring decision-making based on visual contexts, such as video surveillance, self-driving cars, medical image analysis, or robotics. After Krizhevsky *et al.* [1] winning the ILSVRC 2012 contest with AlexNet, Deep Learning (DL) has become the state-of-the-art in computer vision tasks and all subsequent models that won the contest were also Convolutional Neural Networks (CNN). Roughly speaking, the deeper these networks are, the better their performance, albeit at the expanse of overparameterization [2].

Theoretically, models without unimportant parameters may provide better generalization, requiring few training samples to learn the function that maps the input to the desired output and have improved learning speed. The various possible ways of combining convolutional filters with different sizes and many other details allow the creation of models with fewer weights and less computation, as the work of He *et al.* [3], Szegedy *et al.* [4], and Sandler *et al.* [5].

The high computational cost of these models require the use of graphics cards to accelerate the processing, which is not available in many automation scenarios that require the use of constrained hardware. Consequently, there is a growing area in the literature aiming to reduce these models in different aspects, such as reduction of the number of operations, storage size, or energy consumption, depending on the need of the problem at hand.

There are many different forms of model compression with different purposes in the literature. In pruning, for example, a large model is created and different techniques are used to remove unnecessary parameters [6]–[9]. To optimize the storage size reduction, it is also possible to share parameters, grouping them with non-supervised learning [7]. In knowledge distillation, some studies create a smaller version of a good model and try to use the large model to augment the smaller model performance [10]–[12]. In Neural Architecture Search (NAS), the goal is to automate the architecture creation, aiming at a certain number of parameters, inference time, and latency reduction [13]–[16], allowing the deployment on mobile devices.

Despite of the existence of several approaches for model compression, it remains an open question which method(s) provides reasonable trade-offs regarding model complexity and predictive performance. The literature is not uniform in terms of datasets, models, and data augmentation techniques, and to the best of our knowledge, no work to date attempts at comparing substantially different strategies, such as neural architectural search and pruning for example, in a controlled scenario. For that reason, this work aims to provide a thorough comparative study over the performance, compression ratio, storage size, and computational cost reduction of some of the most relevant methods in the model compression literature, focusing on the object detection task within computer vision.

II. Related Work

There are many kinds of model compression methods in the literature, each of which with its goals and peculiarities. Based on the work of Cheng *et al.* [17], Wang *et al.* [18], Choudhary *et al.* [19], Salvi *et al.* [20], and Agarwal *et al.* [21], we group the existing model compression work into the following categories:

- **Parameter Pruning and Sharing**: removal of redundant and uncritical parameters, either in a structured manner (*e.g.,* removing neurons or layers), or in an unstructured manner (removing connections);
- **Quantization**: reduction of the model storage size by reducing the precision of machine parameters representation;
- **Low-Rank Factorization (LRF)**: decomposition of matrices to estimate the informative parameters of the neural network;

- **Transferred/Compact Convolutional Filters**: designing special structures of convolutional filters to reduce the parameter space;
- **Neural Architecture Search (NAS)**: automation of the architecture engineering for deep neural networks. A search strategy generates and trains a network, whose performance guides the search strategy to create another model. The search can be performed by a variety of methods, such as random search, Bayesian Optimization, Evolutionary Algorithm, gradient-based search, or even Reinforcement Learning [22];
- **Knowledge Distillation (KD)**: distillation of a model by training a more compact neural network to reproduce the output of a larger net.

The studies from Cheng *et al.* [17] and Shiming [23] bring theoretical explanations and discussions on pruning, quantization, LRF, transferred convolutional filters, and KD, but do not provide an experimental analysis involving the approaches. Shiming [23] concludes that there is little work on object detection, tracking, and other computer vision tasks. Neither of them approach NAS as a tool for model compression. In [24], the authors relate general constraints and trade-offs in embedded ML and DL, such as reducing latency, increasing reliability, power versus cost, privacy, and security. They explain and recommend which scenarios to use CPU based on RISC or CISC architectures, but they also lack any experimental comparison among the methods.

In [19], the authors survey over LRF, KD, quantization, pruning, and efficient architectures. They bring some practical comparisons between the approaches in isolated cases. *E.g.,* first, there is an analysis showing the parameter reduction, the top-1 and top-5 classification errors of an AlexNet, in a comparison of pruning, pruning with quantization, and LRF, and with a VGG-16, comparing pruning, pruning with quantization, quantization, and LRF. In both cases, they do not analyze neither KD nor efficient architectures. Moreover, their comparison only reported the results of the referenced authors. Thus, there is no way of ensuring that the same training scheme was actually performed, damaging the confidence on the results. Next, they compared the number of parameters, latency, and top-1 classification error on ImageNet of models with efficient architectures, such as SqueezeNet and MobileNet, but without mentioning approaches such as pruning or LRF. Finally, the authors presented a comparison of their own results with an AlexNet on CIFAR10 and MNIST, but they only evaluated pruning and quantization. Hence, the reader cannot tell which method (or set of methods) provides the best trade-off between compression and accuracy.

Wang *et al.* [18] surveyed the following model compression approaches over hardware specific for DL: quantization, pruning, parameter sharing, LRF, and activation approximation. One of the experiments show the throughput (classifications produced per second or classification rate) of an AlexNet over ImageNet versus weight precision versus activation precision in two different hardware with fixed-point representation for the weights and activation function. This is an interesting comparison to evaluate the trade-off between precision and throughput. However, it is only useful for quantization approaches. Another comparison shows the top-1 error rates versus compression rate for implementations of AlexNet trained over ImageNet. Once again, the results that are reported are those of the original references, which do not implement the same training protocol and data augmentation. This comparison analyzes quantization, pruning, weight sharing, and LRF, and thus does not analyze approaches such as NAS or KD. Finally, there is a trade-off comparison between compression and accuracy, but only for quantization approaches in the classification task.

III. Materials and Methods

In this section, we describe in detail the methodology we employ for a fair comparison among different model compression techniques in reducing models for object detection.

A. Models, Datasets, and Training Scheme

We make use of two models as a baselines. The first one, YOLOv3, is the third improvement of YOLO created by Redmon and Farhadi [25], which is a one-shot object detector that performs the entire inference in a single step, up-sampling features and then predicting at three scales, using nine anchor boxes. We use this model as our performance baseline — a compressed model should ideally achieve performance as similar as possible to YOLOv3. The second baseline is YOLOv3-Tiny, which is a handmade reduced version from YOLOv3 that we use as our baseline in terms of compression.

Due to time and hardware constraints, we evaluate all models in two medium-sized datasets. The first dataset is PASCAL VOC 2012 for training/validation and PASCAL VOC 2007 for test [26], which contains 20 classes and $5,717$, $5,823$, and $4,952$ images for training, validation, and test, respectively. The second dataset is ExDark [27], a dataset with exclusively low-light images captured in visible light only, containing $3,000$, $1,800$, and $2,563$ images for training, validation, and test, respectively.

All models we train are based on the YOLOv3 implementation from Jocher *et al.* [28], which performs a smart bias initialization [29] and uses a cosine learning rate decay [30], both for training stability. The GIoU [31] is used for bounding box regression loss, while the remaining loss functions are the same from Redmon and Farhadi [25]. Regarding data augmentation, the images have a shape ranging from 288×288 up to 640×640, and they are built as a mosaic with random crops of four random images with random HSV color jitter, and random horizontal flips. In general, we use SGD with a learning rate $\eta = 0.01$ for YOLOv3 and YOLOv3-Tiny, and $\eta = 0.001$ for YOLO Nano and YOLOv3-Mobile, with momentum $\alpha = 0.937$, and ℓ_2 weight decay with $\lambda = 4.84e-4$, training for 300 epochs with batch size $= 64$.

B. Compression Methods

In the **Parameter Pruning** category, we evaluate two techniques. The first one is Lottery Tickets Hypothesis (LTH) from

Frankle *et al.* [32], which is an iterative non-structured pruning strategy based on the magnitude of the parameter values. In LTH, a layer/model inference is written as $\hat{\mathbb{Y}} \leftarrow \phi(\mathcal{W} \odot \mathcal{M}, \mathbb{X})$, where $\mathbb{X}$ is the input vector, $\mathcal{W}$ is the weight matrix, $\phi()$ is the activation function, and $\mathcal{M}$ is a binary mask, with same shape as $\mathcal{W}$, that decides whether keeping or pruning weights. The mask is initialized with ones and the training is fully-performed, in a cycle known as iteration. In the end, the ρ-th smallest magnitude parameters — either layer-wise or globally — are removed, setting as zero on the mask. The parameters are re-initialized at a back-up generated during training and a new iteration is performed, and it goes on iteratively until the number of remaining parameters reach a desirable compression ratio c, as presented in Algorithm 1.

Algorithm 1 Iterative Magnitude Pruning (IMP) with rewinding to iteration k. Adapted from [32].

```
Require: weight matrix $\mathcal{W}$, mask $\mathcal{M}$, pruning rate $\rho$, backup epoch $k$, total of epochs $n\_epochs$, desired compression $c$, input $\mathbb{X}$, labels $\mathbb{Y}$
1: $random\_initialization(\mathcal{W})$
2: $initialize\_with\_ones(\mathcal{M})$
3: $remaining\_weights \leftarrow count\_parameters(\mathcal{M})$
4: while $remaining\_weights > c$ do
5:     for $epoch \in n\_epochs$ do
6:         if $epoch = k$ then
7:             $\mathcal{W}_k \leftarrow \mathcal{W}$
8:         end if
9:         $\mathcal{W} \leftarrow \mathcal{W} \odot \mathcal{M}$
10:        $\hat{\mathbb{Y}} \leftarrow \varphi(\mathcal{W}, \mathbb{X})$
11:        $\min_{\mathcal{W}} \mathcal{L}(\mathbb{Y}, \hat{\mathbb{Y}})$
12:    end for
13:    $indexes \leftarrow find\_smallest\_values(|\mathcal{W}|, \rho)$
14:    $\mathcal{M}[indexes] \leftarrow 0$
15:    $\mathcal{W} \leftarrow \mathcal{W}_k$
16:    $remaining\_weights \leftarrow count\_parameters(\mathcal{M})$
17: end while
18: $\mathcal{W} \leftarrow \mathcal{W} \odot \mathcal{M}$
19: $train()$
```

The second pruning technique we evaluate is called Continuous Sparsification (CS) from Savarese *et al.* [33]. In this method, the mask $\mathcal{M}$ is re-parameterized over the soft-mask $\mathcal{S} : \mathbb{R}^D$ with $\hat{\mathcal{M}} \leftarrow \sigma(\beta\mathcal{S})$, and SGD is used to continuously learn it as an ℓ_1 regularization problem, where σ is the sigmoid function and β is a temperature value. The higher the β, the steeper is the sigmoid function, closer to binary values. β is increased in equal steps during training from 1 to 200 and is reinitialized after each iteration [33]. After the last iteration, the function $binarize()$ sets all positive $\mathcal{S}$ values to 1, and 0 the remaining. The last training iteration is performed to fine-tune only the model weights, freezing the soft-mask $\mathcal{S}$, as presented in Algorithm 2.

Due to time constraints, we perform LTH both globally and locally using Algorithm 1 with one iteration and a pruning rate ρ of 90%. In CS, we use one iteration of 300 epochs and

Algorithm 2 Continuous Sparsification. Source: Adapted from [33]

```
Require: weight matrix $\mathcal{W}$, soft-mask $\mathcal{S}$, initial value $\mathcal{S}_0$, $\beta$ increment,initial temperature $\beta_0$, backup epoch $k$, total of epochs $n\_epochs$, total of iterations $n\_iterations$, input $\mathbb{X}$, labels $\mathbb{Y}$
1: $random\_initialization(\mathcal{W})$
2: $initialize\_with\_constant(\mathcal{S}, \mathcal{S}_0)$
3: for iteration $\in n\_iterations$ do
4:     for $epoch \in n\_epochs$ do
5:         if $epoch = k$ and $iteration = 0$ then
6:             $\mathcal{W}_k \leftarrow \mathcal{W}$
7:         end if
8:         $\hat{\mathcal{M}} \leftarrow \sigma(\beta \times \mathcal{S})$
9:         $\mathcal{W} \leftarrow \mathcal{W} \odot \hat{\mathcal{M}}$
10:        $\hat{\mathbb{Y}} \leftarrow \varphi(\mathcal{W}, \mathbb{X})$
11:        $\min_{\substack{\mathcal{W}\\ \mathcal{S}}} \mathcal{L}(\mathbb{Y}, \hat{\mathbb{Y}}) + \lambda \cdot \|\hat{\mathcal{M}}\|_1$
12:        $\beta \leftarrow \beta + i$
13:    end for
14:    $\mathcal{S} \leftarrow \min(\beta\mathcal{S}, \mathcal{S}_0); \beta \leftarrow \beta_0; \mathcal{W} \leftarrow \mathcal{W}_k$
15: end for
16: $\mathcal{W} \leftarrow \mathcal{W} \odot binarize(\mathcal{S})$
17: $train()$
```

another approach with three iterations of 100 epochs, with the last training iteration of 300 epochs for both. Thus, there are a total of 300 epochs to build the mask in both cases, as in LTH. We initialize $\mathcal{S}_0 = -0.1$, trying to perform the same pruning rate ρ as in LTH, using $\lambda = 1e-5$ and a mask learning rate of 0.1 as in Savarese *et al.* [33]. For both LTH and CS, the models are re-initialized to epoch 10.

Current DL frameworks allow sparse tensors, storing a list containing the floating-point values and another list containing the respective indexes as unsigned long values. However, they do not allow sparse convolutional operations. Thus, for computing the real storage size and MACs reduction of the pruning approaches, we save the pruned parameters as a $2D$ sparse tensor and forward it as a matrix multiplication between a sparse tensor (the weights) and a dense (the input reshaped to a $2D$ matrix). Hence, we can achieve an effective model reduction, which is not only theoretical. Figure 1 shows an example of the forward pass of a convolutional filter as a matrix multiplication.

In terms of the **Neural Architecture Search** (NAS) approach, we use the YOLO Nano model from Wong *et al.* [13], a reduced version of YOLOv3 based on machine-driven exploration over human design prototyping and the YOLO-family design. It was created as a compressed model for object detection in constrained scenarios such as embedded devices. The main differences between YOLO Nano and YOLOv3 are: the activation functions, which in YOLO Nano are $ReLU6$ and on YOLO$v3$ are $LeakyReLU$; the reduced number of layers; the replacing of the basic blocks from YOLOv3 by efficient blocks using 1×1 convolutions to reduce or increase

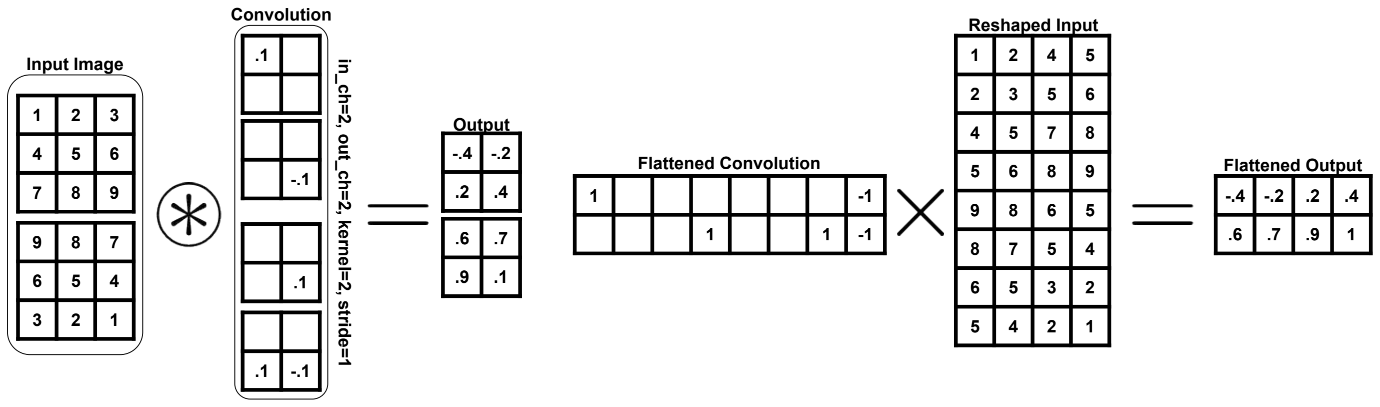

Fig. 1. Example of the forward pass of a convolutional filter as matrix multiplication between a sparse and a dense matrix, with zero padding.

the number of channels and Depth-Wise 3×3 convolutions to reduce the number of parameters; and a fully-connected attention layer.

For the approach in the **Efficient Convolution** category, we built the YOLOv3-Mobile model, replacing all the basic blocks from YOLOv3 by the blocks of MobileNetV3 from Howard *et al.* [14]. MobileNets are a family of CNN architectures built to focus on parameter and latency reductions and speed up the inference time. Its basic blocks are used also in other architectures, such on MnasNet [15] and EfficientNet [16], and its basic blocks also use Depth-Wise convolutions and 1×1 convolutions to reduce the number of channels, as in YOLO Nano. There are two variants of this basic block, using $ReLU$ or $HardSwish$ as activation functions and using or not the Squeeze-and-Excite module. Generally, these variations on MobileNetV3 occur based on the stride, and we keep the same logic in our YOLOv3-Mobile model, whose basic blocks are presented in Figure 2.

In the **Knowledge Distillation** category, we evaluate two techniques. The first one is a classical KD method from

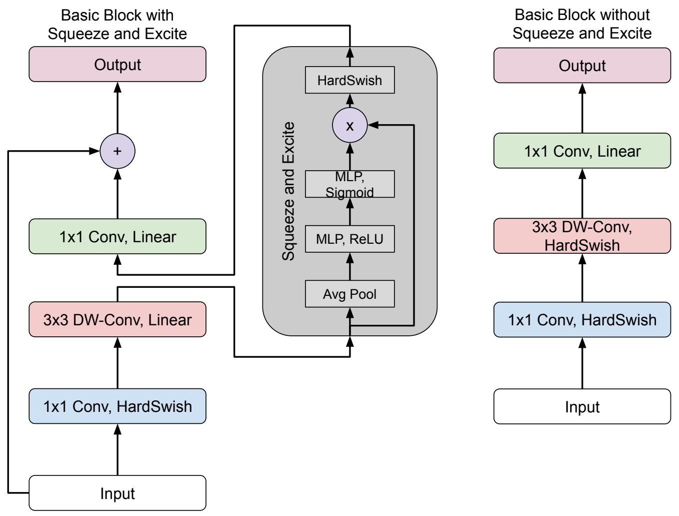

Fig. 2. MobileNetV3 basic blocks. DW-Conv means Depth-Wise Convolution.

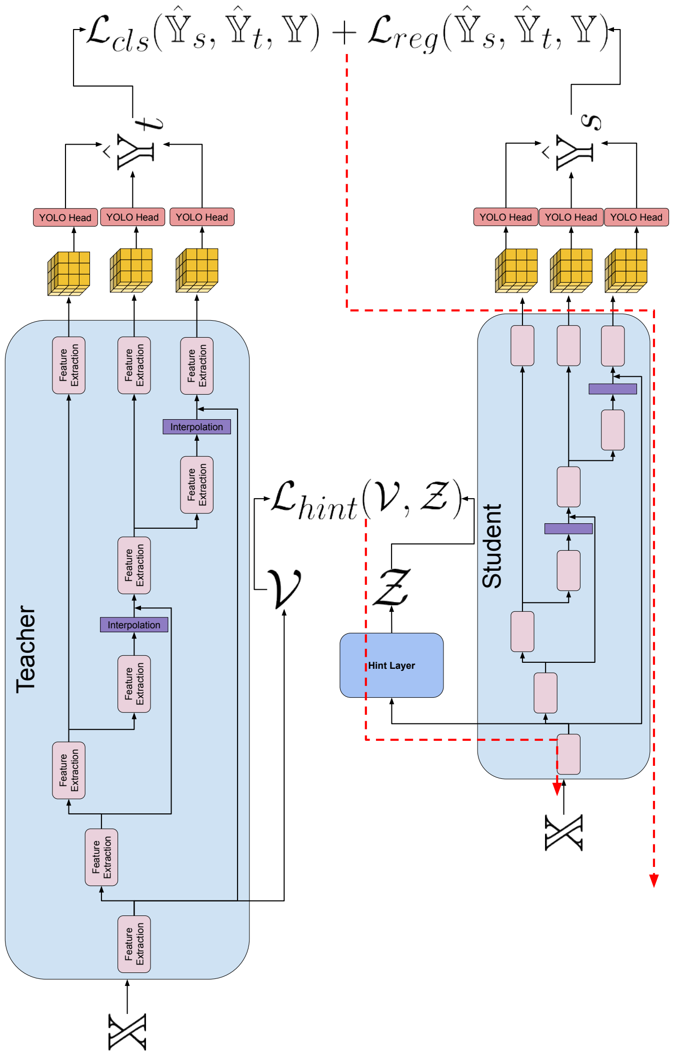
$\mathcal{L}_{cls}(\hat{\mathbb{Y}}_s, \hat{\mathbb{Y}}_t, \mathbb{Y}) + \mathcal{L}_{reg}(\hat{\mathbb{Y}}_s, \hat{\mathbb{Y}}_t, \mathbb{Y})$
$\hat{\mathbb{Y}}_t$
$\hat{\mathbb{Y}}_s$
YOLO Head
Feature Extraction
Interpolation
Teacher
Student
$\mathcal{L}_{hint}(\mathcal{V}, \mathcal{Z})$
$\mathcal{V}$
$\mathcal{Z}$
Hint Layer
$\mathbb{X}$

Fig. 3. Classical KD Approach. Both models are YOLO architectures

Guobin *et al.* [10], which trains a student model to learn with the ground-truth and to mimic the teacher outputs and intermediate features. In this approach, the classification loss is rewritten as $\mathcal{L}_{cls} = \mu\mathcal{L}_{hard}(\mathbb{Y}, \hat{\mathbb{Y}}_s) + (1-\mu)\mathcal{L}_{soft}(\hat{\mathbb{Y}}_t, \hat{\mathbb{Y}}_s)$, where $\mathcal{L}_{hard}$ is the default classification loss between the student inference and the ground-truth, $\mathcal{L}_{soft}$ is a classification loss between the student inference and teacher inference, with μ balancing the two losses. With preliminary experiments, we set $\mu = 0.7$. The regression loss is rewritten as $\mathcal{L}_{reg} = \ell_1(\mathbb{Y}, \hat{\mathbb{Y}}_s) + \nu\mathcal{L}_{tb}(\mathbb{Y}, \hat{\mathbb{Y}}_t, \hat{\mathbb{Y}}_s)$, where ν balances the second term, with $\nu = 0.5$ as default value in Guobing *et al.* [10] and $\mathcal{L}_{tb}$ is the teacher-bounded regression loss, which means the default regression loss between the student inference and ground-truth if this value plus a margin is higher than the loss between the teacher inference and the ground-truth. Finally, there is the hint loss $\mathcal{L}_{hint} = \ell_1 |\mathcal{V}, \mathcal{Z}|$, where $\mathcal{V}$ and $\mathcal{Z}$ are hidden-layer features from teacher and student, respectively. To match the number of channels, there is the Hint Layer, a 1×1 convolutional layer that receives the features from the student and outputs with the same channels as the teacher features. Figure 3 shows that approach adapted to YOLOv3. We create a sub-set and perform preliminary experiments to define the best hint layers combination.

The second KD technique is based on Generative Adversarial Networks (GANs) from the work of Wang *et al.* [12]. Their approach trains the student model in the first $\frac{2}{3}$ of epochs as a generator $\mathcal{G}$ from a GAN, and the last $\frac{1}{3}$ of epochs as an object detector. The features generated by the teacher is the real input set $\mathbb{X}$, and the features generated by the student layer is the fake input set $\hat{\mathbb{X}}$. There are discriminator models $\mathcal{D}$ trying to distinguish between $\mathbb{X}$ and $\hat{\mathbb{X}}$. Thus, via adversarial training, the student learns to generate features by imitating the teacher features. For KD GAN, hint layers are not allowed, thus the teacher and student features need to have the same volume. In our scenario, this happens with the features from the last layer of each branch. There is one $\mathcal{D}$ for each pair of teacher/student features, trained with SGD using $\eta = 0.01$ and a Binary Cross Entropy Loss. The KD GAN is presented in Figure 4.

For both KD techniques, we replace the student activation functions (ReLU6, ReLU, and Hard-Swish) by LeakyReLU, which is the teacher activation function. As the student task is to imitate the teacher, including its intermediate features, these features need to be in the same domain that the teacher's features. This replacement is necessary since ReLU and ReLU6 output only positive values. Hard-Swish outputs some negative values, but only while $x \in (-0.3, 0)$. In contrast, LeakyReLU outputs negative values for any negative input value. For the teacher, we use the best YOLOv3 model, based on its validation mAP.

C. Evaluation Metrics

We evaluate all models according to the following metrics:

- Mean Average Precision (mAP): it is the default metric to evaluate the performance of an object detector;
- Number of parameters;
- Storage size: recall that pruned models are saved as sparse tensors so we can have an effective storage size reduction. For each parameter, there is a list containing the indexes of the parameter location in the original tensor. Thus, the storage size for pruned models is not proportional to the parameter reduction;
- Multiply–Accumulate Operation (MAC): similarly to FLOPS, it gives a sense on the inference effort. *E.g.*, a convolutional layer with a 3×3 kernel ($K = 3^2$), with 3 input channels ($C_{in} = 3$) and 1 output ($C_{out} = 1$) with stride 1 and no padding, when receiving a $5 \times 5 \times 3$ input volume, convolves 9 times ($steps = 9$), since the output will have a 3×3 resolution. Thus, there are 9 steps multiplying kernel at the 3 channels, resulting on $steps \times K \times C_{in} \times C_{out} = 243$ MACs. For the pruned models, the number of MACs is equal to the number of alive parameters times the number of columns from the image reshaped as a $2D$ matrix.

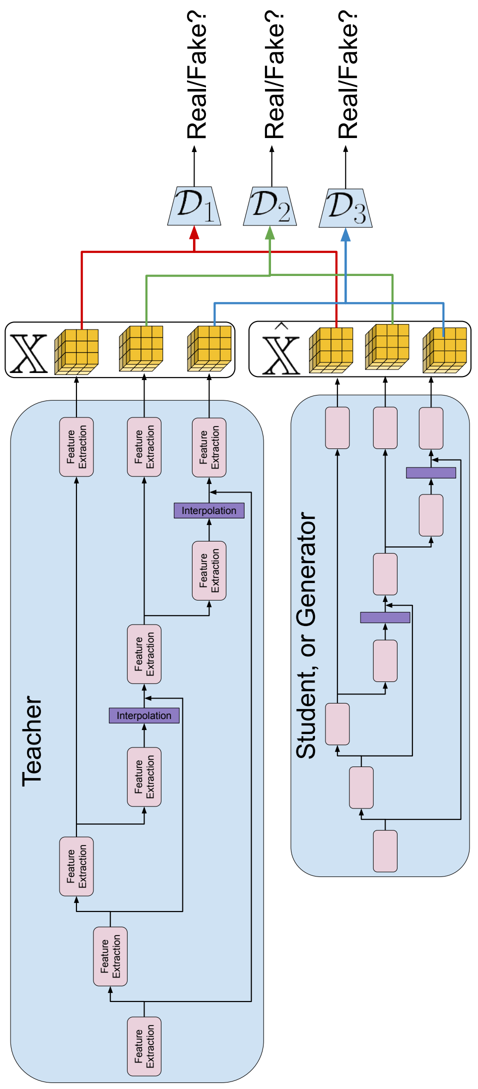

Fig. 4. KD based on GAN

IV. Results

Looking at Tables I and II, the best models regarding mAP are the original YOLOv3 and the models pruned by LTH. Both

LTH with global and local pruning outperforms the original model, with a virtual tie between YOLOv3 default training and LTH local. These results show the effectiveness of both global and local LTH on removing unnecessary parameters, with 2.45% and 0.26% more mAP than default training on PASCAL VOC, and 4.18% and 1.79% more mAP on ExDark.

Global pruning of LTH outperforms local pruning because the latter forces an equal pruning at each layer, while global pruning is more flexible in which locations to remove the parameters. Thus, local pruning has a higher probability of removing more sensitive parameters. On the other hand, layer-wise equal pruning generates, in general, more sparse layers than in global pruning, which favors MAC reduction. Local pruning on PASCAL VOC results in 10.57% of MACs of the original model against 26.79% from global pruning, which is more than double. On Exdark, there are 10.49% of MACs for local pruning and 29.47% for global pruning. Since the storage size between them is the same, 128.26 MB or 50.23% of the original model on PASCAL, and 118.1 MB or 50.20% on ExDark, and the difference in mAP is minimal, it seems local pruning presents the best trade-off overall.

CS generates a more aggressive pruning, being the fourth-best mAP model on PASCAL VOC (one iteration approach), with 0.442 mAP or 80.67% of the original YOLOv3, and the fifth-best mAP model on ExDark with 0.294 mAP or 64.93% of the original model. There is a larger difference between CS with one and three iterations in all metrics and on both datasets. Since the three-iterations approach has 100 epochs per iteration, β increases quickly from the first to the last step, going from 1 until 200, as in Savarese *et al.* [33]. Thus, the derivative of the sigmoidal function vanishes fast, and the model is less capable of learning.

It is important to highlight that in both CS methods, the pruning was more aggressive than in LTH, and consequently, the MACs reduction is greater and storage size is smaller. On PASCAL VOC, it kept only 1.20% of the parameters, 3.47% of MACs, and 4.93% of storage size in the one-iteration approach, against 0.68% of parameters, 1.88% of MACs, and 2.35% of storage size in the three-iterations approach. On ExDark, it kept 0.85% of parameters, 2.87% of MACs, and 3.48% of storage size in the one-iteration approach, against 0.47% of parameters, 1.54% of MACs, and 1.57% of storage size in the three-iterations approach. Both CS approaches present the best reduction of parameters and MACs on both datasets. The three-iterations also provides the smallest storage size on PASCAL, and both provide the smallest storage size on ExDark.

Between the lightweight models, YOLO Nano, YOLOv3-Mobile, and YOLOv3-Tiny, we can see that YOLO Nano generally is the best choice: on PASCAL VOC, it slightly outperforms YOLOv3-Tiny, having 0.385 and 0.379 of mAP, respectively. On ExDark, it has a considerably smaller mAP, with 0.242 and 0.287, respectively. On PASCAL VOC, it has 4.69% of the YOLOv3 parameters, against 14.14% from YOLOv3-Tiny, and on ExDark, 4.66% of parameters against 4.12%; it is the lightest model, requiring 11.38% MB on PASCAL VOC and 11.31 MB on ExDark; and it is the second model with smallest number of MACs. On PASCAL VOC, it has 6.34% of the YOLOv3 MACs against 4.33% from YOLOv3-Mobile, who failed to learn in both datasets and provided insignificant mAP. On Exdark, there are 6.32% of MACs for YOLO Nano and 4.32% for YOLOv3-Mobile.

In Tables I and I, KD fts means the classical KD approach from Guobin *et al.* [10], where the following number indicates the i-th teacher layer used to hint; and KD GAN means the KD GAN-based approach from Wang *et al.* [12]. These tables show that both KD approaches have improved the student's mAP performance. On PASCAL VOC, YOLO Nano on default training achieves 0.385 of mAP or 70.33% of YOLOv3 performance. With KD, it improves to 0.395 — with a technical tie against the default training — (or 72.16%), 0.408 (or 74.50%), and 0.421 (or 76.87%) using KD GAN, KD fts 36, 61, and KD fts 79, respectively. On ExDark, YOLO Nano with default training achieves 0.242 mAP or 53.57% of the YOLOv3 mAP performance. With KD, it improves to 0.254 (or 56.13%), 0.295 (or 65.10%), and 0.303 (or 67.01%),

TABLE I
RESULTS REGARDING mAP, NUMBER OF PARAMETERS, MACS, AND STORAGE SIZE OF MODELS EVALUATED IN THE PASCAL VOC 2007 TEST SET.

Model	**Training**	**mAP**	**Final Params**	**MACs**	**Storage (MB)**
YOLOv3-Tiny	Default	0.379 ± 0.003	$8,713,766$	$2,753,665,551$	33.29
YOLOv3	Default	0.547 ± 0.012	$61,626,049$	$32,829,119,167$	235.44
YOLO Nano	Default	0.385 ± 0.007	$2,890,527$	$2,082,423,381$	11.38
YOLOv3-Mobile	Default	0.009 ± 0.008	$4,395,985$	$1,419,864,487$	17.59
YOLOv3	LTH Local	0.549 ± 0.009	$6,331,150 \pm 1$	$3,468,547,347 \pm 278$	118.26
YOLOv3	LTH Global	$\mathbf{0.561 \pm 0.009}$	$6,331,114 \pm 1$	$8,796,051,025 \pm 225,877,824$	118.26
YOLOv3	CS 1 It	0.442 ± 0.010	$740,072 \pm 12,161$	$1,137,839,381 \pm 44,191,983$	11.62 ± 0.23
YOLOv3	CS 3 It	0.316 ± 0.015	$\mathbf{421,721 \pm 3,544}$	$\mathbf{618,724,616 \pm 20,611,379}$	$\mathbf{5.54 \pm 0.07}$
YOLO Nano$_{leaky}$	KD fts 79	0.421 ± 0.007	$2,890,527$	$2,098,305,681$	11.38
YOLO Nano$_{leaky}$	KD fts 36, 61	0.408 ± 0.008	$2,890,527$	$2,098,305,681$	11.38
YOLO Mobile$_{leaky}$	KD fts 91	0.253 ± 0.023	$4,395,985$	$1,458,910,247$	17.59
YOLO Mobile$_{leaky}$	KD fts 36, 91	0.244 ± 0.010	$4,395,985$	$1,458,910,247$	17.59
YOLO Nano$_{leaky}$	KD GAN	0.395 ± 0.012	$2,890,527$	$2,098,305,681$	11.38
YOLO Mobile$_{leaky}$	KD GAN	0.311 ± 0.006	$4,395,985$	$1,458,910,247$	17.59

TABLE II
RESULTS REGARDING mAP, NUMBER OF PARAMETERS, MACS, AND STORAGE SIZE OF MODELS EVALUATED IN THE EXDARK TEST SET.

Model	Training	mAP	Final Params	MAC	Storage (MB)
YOLOv3-Tiny	Default	0.287 ± 0.020	$8,695,286$	$2,747,415,255$	33.22
YOLOv3	Default	0.453 ± 0.017	$61,582,969$	$32,799,960,583$	235.27
YOLO Nano	Default	0.242 ± 0.013	$2,872,743$	$2,071,460,013$	11.31
YOLOv3-Mobile	Default	0.0003 ± 0.0001	$4,390,537$	$1,416,145,135$	17.57
YOLOv3	LTH Local	0.461 ± 0.012	$6,288,070 \pm 1$	$3,439,388,763 \pm 278$	118.1
YOLOv3	LTH Global	$\mathbf{0.471 \pm 0.018}$	$6,288,035 \pm 1$	$9,665,082,014 \pm 288,425,550$	118.09
YOLOv3	CS 1 It	0.294 ± 0.012	$525,823 \pm 7,684$	$941,520,024 \pm 58,158,009$	8.19 ± 0.15
YOLOv3	CS 3 It	0.139 ± 0.004	$\mathbf{290,746 \pm 1,638}$	$\mathbf{505,248,788 \pm 15,650,702}$	$\mathbf{3.70 \pm 0.03}$
YOLO Nano$_{leaky}$	KD fts 79	0.303 ± 0.008	$2,872,743$	$2,087,342,313$	11.31
YOLO Nano$_{leaky}$	KD fts 61, 91	0.295 ± 0.010	$2,872,743$	$2,087,342,313$	11.31
YOLO Mobile$_{leaky}$	KD fts 91	0.113 ± 0.021	$4,390,537$	$1,455,190,895$	17.57
YOLO Mobile$_{leaky}$	KD fts 36, 91	0.107 ± 0.005	$4,390,537$	$1,455,190,895$	17.57
YOLO Nano$_{leaky}$	KD GAN	0.254 ± 0.007	$2,872,743$	$2,087,342,313$	11.31
YOLO Mobile$_{leaky}$	KD GAN	0.157 ± 0.005	$4,390,537$	$1,455,190,895$	17.57

with KD GAN, KD fts $61, 91$, and KD fts 79, respectively.

KD on YOLOv3-Mobile brings the most impressive results: it changed the performance on PASCAL VOC from default training from 0.0009 mAP or 1.64% of the YOLOv3 peformance to 0.244 (or 44.50%), 0.253 (or 46.29%), and 0.311 (or 56.86%) with KD fts $36, 91$, KD fts 91, and KD GAN, respectively. On ExDark, it improves the default training from 0.0003 mAP or 0.07% of the YOLOv3 peformance to 0.107 (or 23.62%), 0.113 (or 24.95%), and 0.157 (or 34.66%) with KD fts $36, 91$, KD fts 91, and KD GAN, respectively. However, note that none of the methods was sufficient to outperform YOLO Nano. Moreover, none of the methods was sufficient for YOLOv3-Mobile to outperform YOLOv3-Tiny, which is a manually reduced version from YOLOv3.

For YOLO Nano, the classical KD approach performs better, while for YOLOv3-Mobile, KD GAN performs better. We argue that this difference is due to the domain of the generated features: although the activation functions between student and teacher are the same, there is a big macro-architectural difference between YOLOv3 and YOLO Nano. Although YOLO Nano is a NAS model inspired in the YOLO family, detecting at three scales, branching the generated features in three paths, and classifying the objects using YOLO Head, YOLO Nano contains significantly fewer layers than YOLOv3, in addition to not containing skip connections. On the other hand, YOLOv3-Mobile contains the same macro-architecture. Its difference relies only in the micro-architecture, as the layers are different, but the connections between each other, the amount of layers, and their order are the same from YOLOv3.

V. CONCLUSIONS

In this work, we perform a thorough empirical analysis among several model compression techniques, including Parameter Pruning, Neural Architecture Search, and Knowledge Distillation, for the object detection task. We fix the evaluated model, datasets, and training scheme for a fairer assessment of the performance obtained by each method and the gain in resources they achieve. To the best of our knowledge, this is the first work that compares these three macro approaches, evaluating all of them on all proposed metrics of the literature. We also propose a model reconstruction approach for pruned models that is easy to implement, is independent of the Deep Learning framework or hardware, and allows real computing saving for pruning approaches.

Our results show that pruning approaches provide the highest mAP values among the compression approaches: on PASCAL VOC, LTH can generate models with almost 90% fewer parameters while also outperforming the mAP of the original model by more than 2.45%, while NAS generates a model with 70.33% of the original model performance. On ExDark, LTH outperforms the original model in 4.18%, while YOLO Nano (from NAS) provides only 53.57% of the original performance. Looking for the best trade-off between performance and effectively saved resources, pruning approaches also provide the best results. Using CS with aggressive pruning results in a model with 80.67% of the original model performance in PASCAL VOC but with 98.8% fewer parameters, 96.53% fewer MACs, and 95.07% fewer megabytes to store, while its best competitor, YOLO Nano with KD, has 76.87% of the original model performance, with 95.31% fewer parameters, 93.61% fewer MACs, and 95.17% fewer megabytes. On ExDark, YOLO Nano slightly outperforms CS regarding mAP, with 67.01% and 64.93% respectively, but CS has the advantage on parameters, MACs, and storage size.

As future work, we believe it would be enriching to use these model compression methods over other models, like two-step object detectors. Furthermore, we plan on experimenting with new pruning and KD approaches, *e.g.,* the work of Tanaka *et al.* [9], which performs pruning that does not require prior training to find out which parameters need to be removed. Another example is the work of Wu and Gong [11], which performs a collaborative KD, that is, both teacher and student are trained from scratch together. We also have several ideas to improve the evaluated approaches. For instance, the KD GAN approach can be improved with the Kullback-Leibler (KL)

Divergence, so we can force the student feature generation to be more similar to the teacher, similarly as what happens when the KL Divergence is used in GAN studies for image generation. For YOLOv3-Mobile, maybe a restricted NAS will be capable of improving its performance, freezing the macro-architecture and using the NAS to only decide the number of channels of each block and the use or not of the Squeeze and Excite module.

VI. Acknowledgements

This work was achieved in an project supported by the Brazilian Informatics Law (Law nº 8.248 of 1991) and was developed over Agreement 001/2015 between Pontifícia Universidade Católica do Rio Grande do Sul and the HP Brasil Indústria e Comércio de Equipamentos Eletrônicos Ltda.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[2] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," 2016. [Online]. Available: https://arxiv.org/abs/1605.07678

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition*, June 2016, pp. 770–778.

[4] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conference On Artificial Intelligence*, Feb 2017, pp. 4278–4284. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806/14311

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2018, pp. 4510–4520.

[6] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," 2014. [Online]. Available: https://arxiv.org/pdf/1404.1869.pdf

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015. [Online]. Available: https://arxiv.org/abs/1510.00149

[8] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1803.03635

[9] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020. [Online]. Available: https://arxiv.org/pdf/2006.05467.pdf

[10] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 742–751.

[11] G. Wu and S. Gong, "Peer collaborative learning for online knowledge distillation," 2020. [Online]. Available: https://arxiv.org/pdf/2006.04147.pdf

[12] W. Wang, W. Hong, F. Wang, and J. Yu, "Gan-knowledge distillation for one-stage object detection," *IEEE Access*, vol. 8, pp. 60 719–60 727, Mar 2020.

[13] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung, "Yolo nano: a highly compact you only look once convolutional neural network for object detection," 2019. [Online]. Available: https://arxiv.org/abs/1910.01271

[14] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *International Conference on Computer Vision (ICCV)*, Oct 2019, pp. 1314–1324.

[15] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 2820–2828.

[16] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020. [Online]. Available: https://arxiv.org/pdf/1905.11946.pdf

[17] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1710.09282

[18] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. K. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–39, May 2019. [Online]. Available: https://doi.org/10.1145/3309551

[19] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5113–5155, Oct 2020. [Online]. Available: https://doi.org/10.1007/s10462-020-09816-7

[20] A. A. Salvi and R. C. Barros, "An experimental analysis of model compression techniques for object detection," in *Symposium on Knowledge Discovery, Mining and Learning (KDMiLe)*, Oct 2020, pp. 49–56.

[21] S. Agarwal, J. O. D. Terrail, and F. Jurie, "Recent advances in object detection in the age of deep convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/pdf/1809.03193.pdf

[22] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, Mar 2019.

[23] S. Ge, "Efficient deep learning in network compression and acceleration," in *Digital Systems*. IntechOpen, Nov 2018.

[24] M.-A. A'râbi and V. Schwarz, "General constraints in embedded machine learning and how to overcome them—a survey paper," 2019.

[25] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[26] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun 2010.

[27] Y. P. Loh and C. S. Chan, "Getting to know low-light images with the exclusively dark dataset," *Computer Vision and Image Understanding*, vol. 178, pp. 30–42, Jan 2019.

[28] G. Jocher, guigarfr, perry0418, Ttayu, J. Veitch-Michaelis, G. Bianconi, F. Baltacı, D. Suess, and WannaSeaU, "ultralytics/yolov3: Video Inference, Transfer Learning Improvements," Apr. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2624708

[29] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *The IEEE International Conference on Computer Vision*, Oct 2017, pp. 2980–2988.

[30] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Conference on Computer Vision and Pattern Recognition*, Jun 2019, pp. 558–567.

[31] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, California, USA: Computer Vision Foundation, June 2019, pp. 658–666. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Rezatofighi_Generalized_Intersection_Over_Union_A_Metric_and_a_Loss_for_CVPR_2019_paper.pdf

[32] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Stabilizing the lottery ticket hypothesis," 2019. [Online]. Available: https://arxiv.org/abs/1903.01611

[33] P. Savarese, H. Silva, and M. Maire, "Winning the lottery with continuous sparsification," 2019. [Online]. Available: https://arxiv.org/abs/1912.04427