An Experimental Analysis of Model Compression Techniques for Object Detection

Andrey A. Salvi[1], Rodrigo C. Barros[1]

Pontifícia Universidade Católica do Rio Grande do Sul, Brazil
{andrey.salvi@edu., rodrigo.barros@}pucrs.br

Abstract. Recent research on Convolutional Neural Networks focuses on how to create models with a reduced number of parameters and a smaller storage size while keeping the model's ability to perform its task, allowing the use of the best CNN for automating tasks in limited devices, with reduced processing power, memory, or energy consumption constraints. There are many different approaches in the literature: removing parameters, reduction of the floating-point precision, creating smaller models that mimic larger models, neural architecture search (NAS), etc. With all those possibilities, it is challenging to say which approach provides a better trade-off between model reduction and performance, due to the difference between the approaches, their respective models, the benchmark datasets, or variations in training details. Therefore, this article contributes to the literature by comparing three state-of-the-art model compression approaches to reduce a well-known convolutional approach for object detection, namely YOLOv3. Our experimental analysis shows that it is possible to create a reduced version of YOLOv3 with 90% fewer parameters and still outperform the original model by pruning parameters. We also create models that require only 0.43% of the original model's inference effort.

CCS Concepts: • **Computing methodologies** → **Neural networks**.

Keywords: Deep Learning, Efficient Convolutions, Model Compression, Object Detection, Pruning, YOLOv3

1. INTRODUCTION

Deep neural networks have become state-of-the-art in fields such as computer vision and natural language processing. Commonly, the deeper these networks become, the larger their performance is, albeit at the expanse of overparameterization [Canziani et al. 2016]. However, it also decreases the inference speed and makes its use less viable for applications where memory or computational power are constrained, as in mobile devices, embedded systems, CPU-only devices, energy-constrained applications, etc. Consequently, there are different approaches and purposes in model compression, reducing the operations per second, storage space, or energy consumption, depending on the need for the problem to be addressed.

Although there is quite a lot of work in model compression literature, it remains inconclusive which methods provide a good trade-off 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 different strategies in a controlled scenario. For that reason, our contribution here is in providing an initial comparative study over the performance and the compression ratio of three model compression approaches – two different pruning approaches and a Neural Architectural Search (NAS) approach combined with efficient convolutions – within the task of object detection, focusing on the well-known YOLOv3 [Redmon and Farhadi 2018] model. Our goal is to discover how these approaches impact the performance of the model in terms of Mean Average Precision (mAP) and Multiply–Accumulate (MAC) operation when comparing to

the original YOLOv3, and how small can those models become when comparing to YOLOv3-Tiny, a handmade reduced version from YOLOv3.

2. RELATED WORK

Pruning is a post-processing step after training that removes non-significant/redundant connections/neurons based on gradients or other heuristics. Pruning-based approaches rely on the concept that it is necessary to train large networks to find smaller sub-networks with a performance that tends to be the same as the original model. For instance, Frankle *et al.* [Frankle et al. 2019] iteratively train and remove the weights based on the magnitude of the values and reinitialize them to an intermediate state of the training process. The Savarese *et al.* [Savarese et al. 2019] approach continuously learns which weights to remove based on the back-propagation. Despite the pruning approaches presenting, in general, the smallest models with the best performances, they generally generate sparse matrices, which means one needs to remodel its structure to make use of the real model reduction since operations on sparse matrices within the available frameworks/hardware are not typically optimized.

A second model compression strategy is called **Efficient Convolutions**. For instance, Denton *et al.* [Denton et al. 2014] approximate a 3D tensor (the weights from a convolutional layer) through a minimization process, aiming to reduce the difference between the original tensor and an outer product between three tensors. Li *et al.* [Li et al. 2016] create a multi-biased activation function, changing a convolutional layer with N channels by a convolution of N/b channels. Then, they concatenate the obtained feature maps summed with b different biases, reducing the redundancy of parameters. Iandola *et al.* [Iandola et al. 2016] propose the squeeze layer. They perform a sequence of 1×1 convolutional layers, and its goal is to reduce the number of input channels of the following 3×3 convolutional layers. These approaches surely reduce the storage space and the inference cost and can be easily introduced in any architecture, but also decreasing the model performance.

In **quantization**, the goal is to reduce weight storage. For instance, Han *et al.* [Han et al. 2015] use an 8-bit floating-point representation. Courbariaux *et al.* [Courbariaux et al. 2015], Courbariaux *et al.* [Courbariaux et al. 2016] and Rastegari *et al.* [Rastegari et al. 2016] attempt at using a 1-bit representation of weights. Although quantization approaches can considerably reduce the storage space of the neural network, the accuracy is often considerably reduced when applied on large CNNs [Cheng et al. 2017]; thus, it is common to find studies using quantization combined with other approaches, as in Han *et al.* [Han et al. 2015], Gong *et al.* [Gong et al. 2014] and Tung and Mori [Tung and Mori 2018].

Designing tiny networks from scratch is not a good strategy for achieving performance comparable to larger/wider networks. **Knowledge Distillation** (KD) aims at reducing the gap between tiny and large models. Bucilă *et al.* [Bucilă et al. 2006] propose a shallow network trying to imitate the output of an ensemble of teachers. Hinton *et al.* [Hinton et al. 2015] propose a hint layer to guide the intermediate student feature maps. Romero *et al.* [Romero et al. 2014] propose to distill the knowledge of ensemble composed by manny specialist models. Guobin *et al.* [Chen et al. 2017] create a student of object detection model who has no bounding box regression loss when its outputs outperform the teacher. Although KD approaches can reduce the discrepancy between student and teacher models, sometimes both models must have the same architecture family, since the two models represent the features maps in different domains and sometimes learn very different features [Cheng et al. 2017].

3. MATERIALS AND METHODS

The Lottery Tickets Hypothesis (LTH) is a state-of-the-art Iterative Magnitude Pruning (IMP) method from Frankle and Carbin [Frankle and Carbin 2018], which removes the connections between neurons based on their magnitude values. The method was improved by Frankle *et al.* [Frankle

et al. 2019], reinitializing the model to an intermediate state of training. In LTH, we need to create a binary mask $\mathcal{M} \in \{0,1\}^D$ initialized with ones, for each weight matrix $\mathcal{W}$ in the model. The Hadamard product $\mathcal{W} \odot \mathcal{M}$ before each forward pass will control which connections are alive or dead.

In LTH, we train the model until convergence and execute the pruning. With a pruning rate of $p\%$, we prune the $p\%$ remaining weights with the smallest magnitude, meaning that in the same index, we set the mask $\mathcal{M}$ to 0. After pruning, the weights $\mathcal{W}$ need to be reinitialized to a backup, generated at the k-th epoch, where k needs to be a small value between 0.1% and 7% of the total epochs of a execution. This process is repeated iteratively until the number of killed weights achieve the desired compression rate of c. We can prune in a one-shot manner ($p = c$), or iteratively ($p \neq c$).

Algorithm 1 Iterative Magnitude Pruning (IMP) with rewinding to iteration k

Require: inputs X, labels Y, weight matrix $\mathcal{W}$, mask $\mathcal{W}$, backup epoch k, total of epochs n_epochs, compression rate c, pruning rate p

$random\_initialization(\mathcal{W})$
$initialize\_with\_ones(\mathcal{M})$
$n\_alive\_weights \leftarrow count(\mathcal{M})$
while $n\_alive\_weights > c$ **do**
  for $epoch \in n\_epochs$ **do**
    if $epoch = k$ **then**
      $\mathcal{W}_k \leftarrow \mathcal{W}$
    end if
    $\mathcal{W} \leftarrow \mathcal{W} \odot \mathcal{M}$
    $\hat{Y} \leftarrow \phi(\mathcal{W}(X))$
    $\min_{\mathcal{W}} \mathcal{L}(Y, \hat{Y})$
  end for
  $indexes \leftarrow smallest\_values(|\mathcal{W}|, p)$
  $\mathcal{M}[indexes] \leftarrow 0$
  $\mathcal{W} \leftarrow \mathcal{W}_k$
  $n\_alive\_weights \leftarrow count(\mathcal{M})$
end while
$\mathcal{W} \leftarrow \mathcal{W} \odot \mathcal{M}$
$train()$

Algorithm 2 Continuous Sparsification (CS) with rewinding to iteration k

Require: inputs X, labels Y, weight matrix $\mathcal{W}$, soft-mask $\mathcal{S}$, initial value $\mathcal{S}_0$, initial temperature β_0, backup epoch k, total of epochs n_epochs, total of iterations $n\_iterations$

$random\_initialization(\mathcal{W})$
$initialize\_with\_constant(\mathcal{S}, \mathcal{S}_0)$
for iteration $\in n\_iterations$ **do**
  for $epoch \in n\_epochs$ **do**
    if $epoch = k$**and**$iteration = 0$ **then**
      $\mathcal{W}_k \leftarrow \mathcal{W}$
    end if
    $\hat{\mathcal{M}} \leftarrow \sigma(\beta \times \mathcal{S})$
    $\mathcal{W} \leftarrow \mathcal{W} \odot \hat{\mathcal{M}}$
    $\hat{Y} \leftarrow \phi(\mathcal{W}(X))$
    $\min_{\mathcal{W}, \mathcal{S}} \mathcal{L}(Y, \hat{Y}) + \lambda \cdot \|\sigma(\beta\mathcal{S})\|_1$
    $\beta \leftarrow \beta + i$
  end for
  $\mathcal{S} \leftarrow \min(\beta\mathcal{S}, \mathcal{S}_0); \beta \leftarrow \beta_0; \mathcal{W} \leftarrow \mathcal{W}_k$
end for
$\mathcal{W} \leftarrow \mathcal{W} \odot binarize(\mathcal{S})$
$train()$

In Algorithm 1, $\phi(\mathcal{W}(X))$ represent the model inference with ϕ activation function, X input and $\hat{Y}$ predicted output. $\mathcal{L}$ is the loss function, and the min() function will perform the back-propagation. The $smallest\_values()$ function returns the smallest weights and can search locally, returning the smallest values layer by layer, or globally, analyzing the entire model. The $train()$ function performs a normal train after the last pruning, only with the remaining weights.

The Continuous Sparsification (CS) is an alternative approach to the LTH from Savarese *et al.* [Savarese et al. 2019]. In this method, the SGD continuously learns a deterministic mask as a ℓ_1 regularization problem. As the gradient descendent does not create binary values, the mask $\mathcal{M}$ is re-parameterized over the soft-mask $\mathcal{S} : \mathbb{R}^D$ with $\sigma(\beta \times \mathcal{S})$, where σ is the sigmoid function and β is a temperature value. The higher the β, the steeper is the sigmoid function and closer to binary values when $\beta \to \infty$, however vanishing the gradient and not learning. Thus, β is incremented slowly during training to mitigate the vanishing, and the approach reinitialize it after each iteration. Additionally, we reset the soft-mask on the remaining weights with $\mathcal{S} \leftarrow \min(\beta\mathcal{S}, \mathcal{S}_0)$, which not interferes in the removed ones. In Algorithm 2, the soft-mask is initialized at $\mathcal{S}_0$, which controls the sparsity. The min() function

apply the back-propagation in the weights $\mathcal{W}$ and the ℓ_1 regularization over the approximated mask, symbolized by $\|\sigma(\beta\mathcal{S})\|_1$. After learning the soft-mask, the $binarize()$ function sets to 1 the positive values and 0 otherwise. The final $train()$ will fine-tune the model, freezing the soft-mask $\mathcal{S}$.

Wong *et al.* [Wong et al. 2019] propose YOLO Nano, a reduced version of YOLOv3 as a compressed model for object detection in constrained scenarios such as embedded devices. They create YOLO Nano with NAS over human design prototyping and YOLO-family design. YOLO Nano has a macro-architecture similar to YOLOv3, detecting objects at three different scales. The main differences of YOLO Nano from YOLOv3 are the change of the traditional sequential of convolutions with batch-normalization, Leaky-ReLU function, and residual connections by the Projection-Expansion (PE) module and the Projection-Expansion-Projection (PEP) module. Both modules start and end with a 1×1 convolution. These modules internally augment the number of feature channels, and in the last convolution, it reduces the number of output channels, as in the Squeeze Layer [Iandola et al. 2016], reducing the number of parameters. Another trick used in these modules to reduce the number of parameters is that all 3×3 convolutions are of the Depth-Wise type. For instance, in a traditional convolution over an RGB image, a single convolutional filter has three channels, each convolving its respective input channel, and the outputs are summed, resulting in a feature map with one channel. The depth-wise convolution stacks the outputs, resulting in a feature map with three channels and having only $\frac{1}{3}$ of parameters. In PE and PEP modules, a Batch-Normalization layer and a ReLU6 activation function follow each convolution.

Briefly speaking, Multiply–Accumulate (MAC) is an operation when a machine computes the product of two numbers and adds that product to an accumulator as a linear transformation. We can count, *e.g.*,, N MACs when a convolution operation of N channels performs one step of the convolving operation. Counting MAC operations is a way of measuring computational effort to make an inference from a convolutional network. The higher the number of MACs, the higher the power consumption to carry out a forward, which is undesirable mainly in devices powered by a battery.

Despite the pruning approaches reducing many parameters, it produces sparse matrices, which in practice are the same as standard matrices, but with several zeroed elements. Frameworks support storage optimized for sparse matrices, storing only the alive weights in an array and the corresponding indexes. However, to perform a forward, it is necessary to inflate this structure with zeros creating the non-optimized sparse convolution, which does not reduce the MACs. Since Frankle *et al.* [Frankle et al. 2019] and Savarese *et al.* [Savarese et al. 2019] does not deal with this problem, we propose to reconstruct each convolutional pruned layer trying to reduce the number of MACs. We split each convolution each time we find a convolution filter completely pruned, removing the completely pruned one. Thus, to perform a forward, we forward with each sub-convolution and stack the output features, filling with zeros in the holes so that the generated output feature map is equal to the one generated by the original convolution.

4. EXPERIMENTS AND RESULTS

We use the YOLOv3 implementation from Jocher *et al.* [Jocher et al. 2019]. There is a manual bias initialization as in Lin *et al.* [Lin et al. 2017] and a cosine learning rate decay as in He *et al.* [He et al. 2019] to deal with the background prevalence and class imbalance. We train using Stochastic Gradient Descendent (SGD) with a learning rate of 0.01 for YOLOv3 and Tiny, and 0.001 for YOLO Nano, a momentum of 0.937, weight decay of $4.84e-4$, and a batch size of 64 (default values from Jocher *et al.* [Jocher et al. 2019]). We train with images across multiple scales, from 320×320 until 640×640, building as a mosaic (building an image over random crops over four random images) with random HSV color jitter and random horizontal flips.

We train the models by 300 epochs. In pruning, we use the YOLOv3. We train by 300 epochs for the LTH and perform both global pruning and local pruning in a one-shot of 90% ($p = c = 0.9$) and

then the final training of 300 epochs. For the CS, we use $\mathcal{S}_0 = -0.1$ (trying to perform the same pruning rate as in LTH), a mask learning rate of 0.1 with our cosine learning rate decay, β ranging from 0.1 to 200, and a λ weight regularization of $1e-8$ (default values in Savarese *et al.* [Savarese et al. 2019]). We perform the CS both training 1 iteration of 300 epochs to learn the soft-mask $\mathcal{S}$ as 3 iterations of 100 epochs, totalizing 300 epochs to learn the soft-mask in both cases, and after a final training of 300. Both in LTH as in CS, after the iteration end, we rewind the weights to epoch 10.

Like Redmon and Farhadi [Redmon and Farhadi 2018], we create our anchor boxes with $k-Means$ using as distance metric the Intersection Over Union (IOU). With *train/val* splits from PASCAL VOC 2012 and $k = 9$, the anchor boxes for YOLOv3 and YOLO Nano are:$(26; 31)$, $(43; 84)$, $(81; 171)$, $(103; 68)$, $(145; 267)$, $(180; 135)$, $(247; 325)$, $(362; 178)$, $(412; 346)$. With $k = 6$, the anchor boxes for YOLOv3-Tiny are: $(26; 31)$, $(43; 84)$, $(81; 171)$, $(103; 68)$, $(145; 267)$, $(180; 135)$, $(247; 325)$, $(362; 178)$, $(412; 346)$. We train for 5 times with *train/val* splits from PASCAL VOC 2012 [Everingham et al. 2012] and evaluate on PASCAL VOC 2007 [Everingham et al. 2007] *test* split, which contains $4,952$ images. We evaluate the models computing the mAP 0.5, counting the number of parameters, and counting the MAC operations over a image with size $416 \times 416 \times 3$.

In Table I, we can see the results of all the experiments. The best mAP results are from the original YOLOv3, and the respective model pruned with LTH both by local pruning and global pruning. The pruned YOLOv3 with LTH global has a mean mAP higher than the unpruned model, and it has approximately 9.99% of the unpruned YOLOv3 parameters and 70.71% of the YOLOv3-Tiny.

Table I. Values of mAP at PASCAL VOC 2007 test set and the corresponding number of parameters and MACs.

Model	**Pruning**	**mAP 0.5**	**Nº of parameters**	**MACs(G)**
YOLOv3-Tiny	No	0.393 ± 0.004	$8,707,248$	2.75
YOLOv3	No	0.560 ± 0.007	$61,573,216$	32.83
YOLOv3	LTH - Local	0.552 ± 0.013	$6,157,357$	32.614 ± 0.066
YOLOv3	LTH- Global	$\mathbf{0.567 \pm 0.009}$	$6,157,321$	32.264 ± 0.066
YOLO Nano	No	0.369 ± 0.01	$2,873,026$	2.08
YOLOv3	CS - 1 It	0.457 ± 0.013	$\mathbf{268,800}$	**0.21**
YOLOv3	CS - 3 It	0.335 ± 0.011	$\mathbf{268,800}$	**0.21**

The LTH create models at least 0.139 mAP points higher than YOLOv3-Tiny, 0.154 mAP points higher than YOLO Nano, and 0.067 mAP points higher than the pruned models with CS. However, it is important to note that the models pruned by LTH have around 10% of parameters of the original YOLOv3, YOLO-Tiny has 14.14%, YOLO Nano has 4.66%, and the models pruned by CS have 0.43%. The pruned models with LTH are superior in mAP and are smaller than YOLOv3-Tiny. This behavior shows that it is better to prune with LTH a large model than reduce the number of layers and simplifying the output with fewer anchor boxes as in YOLOv3-Tiny.

Table I shows a previously expected behavior: LTH with global pruning performs better than with local pruning. This behavior happens because local pruning forces model pruning at each network layer, while global pruning forces the pruning procedure on network locations that are less sensitive to the network performance. For instance, in one of the three final layers (which performs the classification and bounding box regression), the network performs a convolution with shape $75 \times 256 \times 1 \times 1$ with $19,200$ parameters. While the local pruning procedure left only $1,920$ parameters alive, each iteration with global pruning leaves a different value, having an average of $14,210$ parameters, or 74.01% the original size, against 10% from local pruning.

To a better analysis, we perform an Analysis of Variance (AOV). Table II shows the results from YOLOv3 with and without LTH. Variation Source is the factor analyzed (LTH local, LTH global and

Table II. Analysis of Variance (AOV) among YOLOv3 models with and without LTH pruning.

Variation Source	Degrees of Freedom	Sum of Squares	Mean Square	F Test
Prune	2	$5.665e-4$	$2.833e-4$	2.783
Residuals	12	$1.2212e-3$	$1.018e-4$	-
Total	14	$1.7877e-3$	$3.851e-4$	-

without LTH), Sum of Squares measures data variation, and Mean Square is the ratio between the Sum of Squares and the Degrees of Freedom.

According to the variance analysis, comparing the values obtained with the values of the F-distribution table of Snedecor, the computed values of the $F\ Test$ do not show significant variance. These results show us that the variations obtained between the executions were consistent among themselves, and the use of both local and global pruning did not impact this behavior. Besides that, these results show us that the difference between the models with and without LTH was not significant, clearly showing that it is preferable to use models with LTH over models without pruning. However, the reduction in MACs of the pruned models by the LTH was minimal. In the best case, the pruned model has 0.124 Giga MACs fewer than the original model. Thus, using these models with our convolution reconstruction approach in cases of battery dependent or CPU-only devices is just as unlikely to the model without pruning.

Looking for Table I we can see that YOLO Nano has not only fewer parameters, with approximately 4.66% of the number of parameters of YOLOv3 and 32.89% of the number of parameters of YOLOv3-Tiny, but also requires fewer effort to make an inference, requiring approximately 6.34% of Giga MACs of the YOLOv3 and 75.73% of the YOLOv3-Tiny. YOLOv3 pruned with CS achieved more impressive results in size and computational effort: these models approximately have just 0.43% of the number of parameters of the unpruned model and 3.09% of parameters of the YOLOv3-Tiny, and requires approximately 0.64% of the Giga MACs of the unpruned model and 7.63% of the YOLOv3-Tiny.

In Table 4, we can see that the pruned models with CS using one iteration have a higher mAP than with three iterations. These models are at least 0.042 mAP points higher than YOLOv3-Tiny, 0.057 higher than YOLO Nano, and 0.097 higher than the YOLOv3 pruned with CS using three iterations. As in the three-iteration approach, each iteration has 100 epochs, β increases quickly from the first to the last step, going from 1 until reaching the ceiling value of 200, as in Savarese *et al.* [Savarese et al. 2019]. Thus, the derivative of the sigmoidal function vanishes fast, and the model learns less.

We also analyze the results between YOLOv3-Tiny, YOLO Nano, and the pruned models with CS in terms of analysis of variance, as we can see in Table III. Here, the Variation Source $Model$ means YOLOv3-Tiny, YOLO-Nano, CS with one iteration and three iterations. Comparing the values obtained with the values of the F-distribution table of Snedecor, the computed values of the $F\ Test$ show a significant variance, that is, the difference between the averages obtained is significant. To ensure the comparison and be sure about which average stands out from the others, we performed the Tukey Test using the same data used to compute the Table III.

Table III. Analysis of Variance (AOV) among the least expensive models.

Variation Source	Degrees of Freedom	Sum of Squares	Mean Square	F Test
Model	3	0.03986	0.013288	129.3
Residuals	16	0.00164	0.000103	-
Total	19	0.0415	0.013391	-

Table IV. Tukey Test among the least expensive models.

Comparison	Difference	Lwr	Upr	P Adj
Nano - CS 3It	0.0342	0.0158	0.0525	0.0003
Tiny - CS 3It	0.0578	0.0394	0.0761	$6e-7$
CS 1It - CS 3It	0.1222	0.1038	0.1405	0.0
Tiny - Nano	0.0236	0.0052	0.0419	0.0098
CS 1It - Nano	0.0880	0.0696	0.1063	0.0
CS 1It - Tiny	0.0644	0.0460	0.0827	$1e-7$

In the Table IV, Comparison shows the two factors compared, Difference shows that the first factor averages N points higher (or lower if negative) than the second factor, Lower and Upper shows the lower and upper difference between the two factors at 95% of the confidence interval, and P Adj means the $p-value$ adjusted for multiple comparisons. Nano and Tiny refer to YOLO Nano and YOLOv3-Tiny, and CS 1It and CS 3It refer to models pruned with CS using 1 and 3 iterations, respectively. As we can see, there is a statistical difference in all the means, once time all the P Adj values are less than 0.05. Thus, we can ensure that the pruned models with CS using three iterations have a significantly higher mAP than the other tested models.

5. CONCLUSION AND FUTURE WORK

This article evaluates some approaches for model compression on YOLOv3 trained at PASCAL VOC 2012 *train/val* sets and evaluated at PASCAL VOC 2007 *test* set. We can found models with similar mAP to YOLOv3 and 90% fewer parameters using pruning from Frankle *et al.* [Frankle et al. 2019] and outperforming the unpruned model on average by 0.007 mAP points. We also found models less expensive than YOLOv3 and YOLOv3-Tiny, with YOLO Nano from Wong *et al.* [Wong et al. 2019] - having 6.34% and 75.73% of the Giga MACs of YOLOv3 and YOLOv3-Tiny, respectively - and using pruning from Savarese *et al.* [Savarese et al. 2019] - with 0.43% and 3.09% of the Giga MACs of YOLOv3 and YOLOv3-Tiny, respectively. However, these approaches cannot generate a model with performance (mAP) similar to YOLOv3 and with an inference effort (MAC) similar to YOLOv3-Tiny.

Now, we intend to train YOLO Nano with Knowledge Distillation approaches, such as KD adapted for object detection by Guobin *et al.* [Chen et al. 2017], which trains a shallower version of Faster R-CNN. Another option is the latest versions of KD as a Generative Adversarial Network (GAN), such as adapting the work of Wang *et al.* [Wang et al. 2018], which uses KD in a GAN framework to train shallow networks for Image Captioning. Another example is the work of Wang *et al.* [Wang et al. 2020], which uses KD in a GAN framework to train SSD, a one-shot object detector competitor of YOLO. These experiments are to find if there is possible to improve YOLO Nano performance (mAP) to a value similar to YOLOv3.

6. ACKNOWLEDGEMENTS

This paper was achieved in cooperation with HP Brasil Indústria e Comércio de Equipamentos Eletrônicos LTDA. using incentives of Brazilian Informatics Law (Law nº 8.2.48 of 1991).

REFERENCES

Bucilă, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, United States of America, pp. 535–541, 2006.

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

Chen, G., Choi, W., Yu, X., Han, T., and Chandraker, M. Learning efficient object detection models with knowledge distillation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, United States of America, pp. 742–751, 2017.

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

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., pp. 3123–3131, 2015.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. https://arxiv.org/abs/1602.02830, 2016.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., pp. 1269–1277, 2014.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

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

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

Gong, Y., Liu, L., Yang, M., and Bourdev, L. Compressing deep convolutional networks using vector quantization. https://arxiv.org/abs/1412.6115, 2014.

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

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. Bag of tricks for image classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, United States of America, pp. 558–567, 2019.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. https://arxiv.org/abs/1503.02531, 2015.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. https://arxiv.org/abs/1602.07360, 2016.

Jocher, G., guigarfr, perry0418, Ttayu, Veitch-Michaelis, J., Bianconi, G., Baltaci, F., Suess, D., and WannaSeaU. ultralytics/yolov3: Video Inference, Transfer Learning Improvements, 2019.

Li, H., Ouyang, W., and Wang, X. Multi-bias non-linear activation in deep neural networks. In *International conference on machine learning*. New York, United States of America, pp. 221–229, 2016.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollar, P. Focal loss for dense object detection. In *The IEEE International Conference on Computer Vision (ICCV)*. Venice, Italy, pp. 2980–2988, 2017.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *The European Conference on Computer Vision (ECCV)*. Cham, Switzerland, pp. 525–542, 2016.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement. https://arxiv.org/abs/1804.02767, 2018.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. Fitnets: Hints for thin deep nets. https://arxiv.org/abs/1412.6550, 2014.

Savarese, P., Silva, H., and Maire, M. Winning the lottery with continuous sparsification. https://arxiv.org/abs/1912.04427, 2019.

Tung, F. and Mori, G. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, United States of America, pp. 7873–7882, 2018.

Wang, W., Hong, W., Wang, F., and Yu, J. Gan-knowledge distillation for one-stage object detection. *IEEE Access* vol. 8, pp. 60719–60727, Mar, 2020.

Wang, X., Zhang, R., Sun, Y., and Qi, J. Kdgan: Knowledge distillation with generative adversarial networks. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., pp. 775–786, 2018.

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