Proceedings of the Seventh International Conference on Research in Intelligent and Computing in Engineering pp. 243–248 ISSN 2300-5963 ACSIS, Vol. 33
DOI: 10.15439/2022R09

GreedySlide: An Efficient Sliding Window for Improving Edge-Object Detectors

1st To Hai Thien
University of Transport Technology
Hanoi, Vietnam
thienth@utt.edu.vn
0000-0002-2099-1863

2nd Tung-Lam Duong
ASI Company
Hanoi, Vietnam
adamduong26111996@gmail.com
0000-0002-9459-4705

3rd Chi-Luan Le
University of Transport Technology
Hanoi, Vietnam
luanlc@utt.edu.vn

***Abstract*—The recent development in deep learning and edge hardware architecture has provided artificial applications with a robust foundation to move into real-life applications and allow a model to inference right on edge. If a well-trained edge object detection (OD) model is acquired, multiple scenarios such as autonomous driving, autonomous hospital management, or a self-shopping cart can be achieved. However, to make a model well-inference on edge, a model needs to be quantized to scale down the size and speed up at inference. This quantization scheme creates a degradation in the model where each layer is restricted to at most lower representations, forcing an output layer only to have fewer options to circle an object. Furthermore, it also limits model generalization where the behavior of the dataset gets cut off each activation layer. We proposed a novel method GreedySlide by sliding window that divides a capture into windows to make an object fits better on the quantization bound to address this problem. Even though the technique sounds simple, it helps increase the number of options for bounding an object and clips the variance that can have by scanning the whole image. Our work has improved an original edge model on its corresponding benchmark by experimenting and increasing the model generalization on other related datasets without retraining the model.**

I. Introduction

Recent developments in artificial intelligence have given researchers a powerful tool for data exploration and analysis in many fields, especially in Computer Vision. Many Computer Vision tasks have been overcome by artificial intelligence models and have reached state-of-the-art results such as classification, detection, and recognition,... To take advantage of this premise, researchers are now making ways to bring these SOTA to edge devices to extend AI to another level. Bringing a model into an edge device can blow away people's concerns about leaking personal data when every piece of information and process stays inside a device, and only their encrypted features can be transferred outside for server computing and analysis. This helps AI operate closer to the human world and turn into personal assistance in many tasks

However, most SOTA models contain a huge trained number of parameters in full precision on servers, containing a huge number of layers and parameters. This incident makes them heavy in computation, and it is not reasonable to run the whole model on any edge devices. While high computation causes the edge device to lose more power, it also prevents incompatible edge hardware from running in real-time inference. Thus, they have to work simultaneously on building edge hardware for a friendlier deep learning configuration and reducing the number of bit inferences on a model that an edge has to handle. Newly edge AI chips such as Coral, Ambient,... typically rely on in-memory or near-memory data flow designs that place the logic and the memory data closer together for faster inference [3]. However, even with a strong hardware configuration, some model still takes quite a long time to generate a good result on the server-side. Therefore, the demand of **Model Compression** i.e finding a good model structure and reducing its size for low computational and relatively small power consumption, is leading in the latest research works. By deploying model compression, a two-fold benefit of minimizing the total number of energy-intensive memory accesses [17] and increasing the inference time due to effectively higher memory bandwidth, reducing the overall latency [12].

Regardless of the promising benefits, model compression has limited the capacity for generalization. In a well-inferencing model that can perform well after a model's compression, weights, biases, and activation values have to be retrained to match the new configuration. This step can be time-consuming and prohibit researchers from exploiting SOTA results that take days and months to complete. However, this is not the biggest problem that model compression can cause to a model. In Kim's work [6], he proposed a position-based scaled gradient as a training optimizer that scales the gradient depending on the position of a weight vector for friendly model compression. While for previous work of [9], [7] and [5], they focus on mimicking activation by mean and variance to represent the distribution of activation in the training dataset. By forcing the model to choose the parameters lower bit that fit with the distribution of weights and biases in a network in a higher precision scheme, it has taken away the uncertainty of a model and made it too robust to the training data's behavior, ie. poor model generalization. The lower the bit range, the less degree of freedom to tune with the parameters, which restricts the search space [8]. In addition, the validation for a deep learning model compression from previous work addresses only the same package's testing data, which can not verify a compressed model's true generalization.

Moreover, when considering bringing model compression to an object detection scheme, the model is more numerically

sensitive than the Image Classification process. For [6] and [2] work of Image Classification, the final result for a model is a certain value which is clamping by softmax distribution between 0 and 1 and pushing its maximum a posteriori on a correct class. While a maximum posterior distribution value does not need to be determined exactly, the bounding boxes represented for object detection have to be fitted with the image's pixel location, especially for lower scale objects in the image.

By naively inspecting an input image for an object detection task, the representation of output, when scaled into a lower range bit, could be insufficient. For an input image that has a size (weight x height) larger than the current bit range, a set of quantization range can naively draw as a 255x255 grid for 8-bit inference. Vertices of a bounding box can fall diversely to fit with the ground truth label as they are assigned to each intersection of the grid, which is the corresponding lower-bit representation. After compression, for example, if a vertice value has not passed entirely to a new intersection on the grid, it will be forced heavily back to the previous intersection, which causes shifting in the entire bounding box. A small shifting pixel may not affect big objects, but it can cause problems for small and medium size objects in the scene (Figure 1), which causes a drop in model performance after a quantization process.

Fig. 1: By naively dividing an image in 255x255 grid quantization range (green lines), vertices q1 and q1' are different by two quantized values. Same for q2 and q2'. The detection between 2 close quantized values can affect differently on the final result based on the different scales of objects in an image

Our paper proposes a new method called GreedySlide, which addresses these problems dynamically within the scope of maintaining a good generalization for the model and easy integration on the edge interface. In GreedySlide, we take the trained model's outputs as the purest components to exploit how well a model explores the hidden data pattern and its restriction in generalization. We will then perform a greedy sliding window policy to scan over the image to obtain sub-location bounding boxes in the image and compare that with the current outputs to select the global bounding boxes for the whole image. Details of the policy will be explained in III. By scanning over the image instead of using a single image for detection, we emphasize better size for lower-scale objects and create a suitable range that matches the model's training size during compression. Therefore, it can bypass the overfitting compression when lowering the bit inference.

II. Background - Related Work

Existing methods for object detection using CNNs can be classified into two-stages and one-stage approaches. In two-stages methods such as FasterRCNN [14], R-FCN [1], Retinanet [10] classification and localization are implemented using two separate steps involving classification and region proposal. In contrast to this, the ones-stage approaches (such as Yolo [13], SSD-MobilenetV2 [11] classify and localize objects in only one step. Generally, one-stage detection models are faster by combining two stages as one, while the accuracy of two-stages models is higher. However, if scale down the accepted accuracy to a smaller intersection of the ground truth and the predicted object (intersection of union (IoU) = 0.5), one-stage models can achieve nearly the same accuracy of the two-stages methods. To make use of a one-stage detection structure for fast and real-time inference on an edge device, researchers [9] [15] [4] have tried to replace the feature extraction part of the one-stage scheme with a smaller model and a roughly same efficiency on the full precision setting.

Small objects detection: However, despite fast inferencing, object detections from the one-stage model usually get problems in detecting small-scale objects as the convolution features of these objects generally disappear in the last layers. Due to this problem, the normal solution is enlarging the input image so that the small object's pixels will increase in the training pictures or applying adding features map of upper layers. While upsampling the image size costs more time of inferencing and hyperparameters volume, the feature-map of upper layers makes the one-stage model behave as the second-stage model [16]. As the original model for the single stage has already struggled to handle small objects, a quantized version of it can not assure better performance and small object detection at the edge remains challenging. In our work, we fused both these ideas together. Instead of upsampling the whole image, we emphasize only the area may contain small objects through sliding windows and by providing a subsequent of the original image, we have naively provided the raw feature map of the model.

III. Proposed GreedySlide Algorithm

As briefly discussed our method focus on the post-training phase rather than bringing it into the training pipeline. By applying at the post-training stage, GreedySlide allows to take advantage of any model inference performance ie. different bit scale and improve model generalization. To perform our GreedySlide Algorithm, we divide our work into three phases: Sliding Windows Detection, Bounding Boxes Suppression and

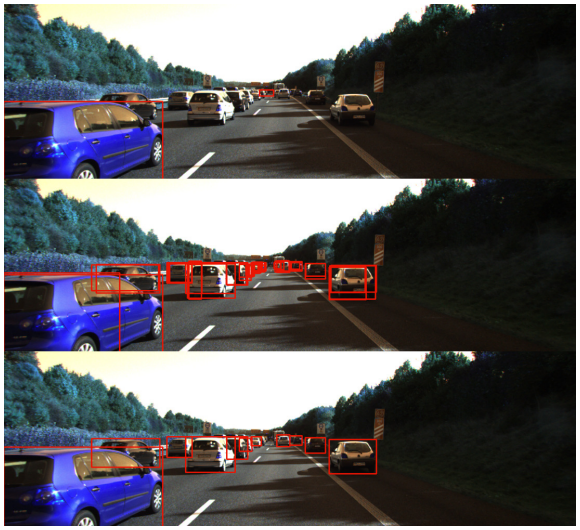

Fig. 2: Top picture is the result of bounding boxes by SSD-MobilenetV2 quantized edge model "occurring shifting detection"; Middle picture is the "partition" bounding boxes after Sliding Windows Detection; Bottom picture is the final bounding boxes after Bounding Boxes Suppression and Greedily Bounding Boxes Selection

Greedily Bounding Boxes Selection. The illustration of this whole pipeline is shown in Figure 2.

A. Sliding Windows Detection

Instead of taking only the whole image as an input for the model, the first phase of the GreedySlide algorithm divided the image into overlapped windows W size (l, l) combined with the original image I for inferencing. There are two main points in this basic approach. First, by overlapping windows on each other by γ threshold, we aim to avoid big objects being sliced into separated parts. Each window through the same ConvNet model only tells what object is detected in them, without knowledge of the whole image. Therefore, we combined the windows with the whole image as a true input for the model. For the second point, the optimal sliding width of the window l should be close to the training size of the model as the resizing do not distort much on the original image, and the image from each window can fit better to the quantized layer's behavior of low-bit range model which can overcome the problem of shifting bounding box for quantization scheme.

$$SLW(W, I, l, \gamma) = W', I' \tag{1}$$

This can be considered batch inference for multi-segment of the same image, which gives the model more elucidated input of the original image and supports multiple resolutions. Each subsequent range of the image will emphasize low-scale objects better as the appearance of those are dynamically bigger in a window. In addition, the window image can represent again the raw feature location of the image, which is useful for the model that has been trained without looking in this data and especially for the quantized model where the feature learning can be saturated.

B. Bounding Boxes Suppression

The reason we called this phase suppression is that there will be a lot of partial bounding boxes coming from big objects that can be scattered in the results. This phase will act as a constrained filter to remove those by greedily remove from the output. In detail, depending on the frequent location of the objects as well as the baseline performance of the model, GreedySlide algorithm takes into account another two threshold numbers to help suppress the bounding boxes. α is designed as the confidence to select a bounding box for each window, and β is the confidence to believe in the baseline model. In general, most object detection models are confident in getting high-scale objects rather than low-scale objects. By selecting correct β we can take out the most confident bounding boxes (global bounding boxes) from the whole image and use it to suppress scattered partial bounding boxes (local bounding boxes) from the windows.

Normally, two bounding boxes are the same if their IOU is high. However, because partial bounding boxes are relatively smaller than the groundtruth bounding boxes, using IOU is insufficient. Thus, we use self-intersection factor (f) to evaluate how strong the local bounding boxes attach to the global bounding boxes. A self-intersection factor f is calculated by the amount of the intersection of a bounding box over its own size, as described in (2).

$$f = \frac{intersection}{bounding\ box\ area} > \epsilon \tag{2}$$

For each local bounding box, it will be compared to global bounding boxes by a matrix relationship between bounding boxes. Each row of the matrix is a self-intersection factor f. Multiple local bounding boxes think to belong to one if f is bigger than ϵ amount. However, the selected ϵ is tricky to obtain a good result. A small ϵ can leave duplicating bounding boxes, while a big ϵ can leave small bounding boxes that scatters around an object due to its low certainty. A good ϵ forces the bounding boxes that are heavily related to a bigger box to be one and avoid wrongly taking close contacted bounding boxes of another objects.

C. Greedily Bounding Boxes Selection

After filtering the partial bounding boxes of the global bounding boxes, there can be the partial bounding boxes of small and medium-scale of objects in among the windows. However, these bounding boxes have no global bounding boxes to suppress them. Therefore, in this phase, the procedure is slightly different from the previous stage. GreedySlide will perform to compare each local bounding box together by the self-intersection-factor f. Then, bounding boxes that are strongly related to each other will be merged as a group G represents for an object. From the group of bounding boxes G, GreedySlide selects a bounding box that represents all of other boxes by select the top vertex *x1,y1* and the bottom

TABLE I: COCO BenchMark of two models (SSD-MobilenetV2, Retina) with its Greedy version in different precision. The benchmark COCO protocol allows to addess the IoU from 0.5 to 0.95 and estimate the performance in terms of recall and precision. The SSD-Mobilenetv2 is conducted on Google Coral Board, while the heavy Retina is measured on Jetson-Nano

	300x300-91 classes-Int8		640x640-80 classes-Fp16		640x640-80 classes-Fp32	
Metrics \ Models	Mobinet SSD	Greedy Mobile SSD	Retina	Greedy Retina	Retina	Greedy Retina
AP(0.5) - (S,M,L)	0.21	**0.24**	0.35	**0.41**	0.39	**0.41**
AP(0.5) - S	0.02	**0.06**	0.11	**0.22**	0.14	**0.21**
AP(0.5) - M	0.13	**0.25**	0.39	**0.45**	0.44	**0.46**
AP(0.5) - L	**0.45**	0.42	**0.55**	0.54	**0.55**	0.54
AR(0.5) - (S,M,L)	0.21	**0.26**	0.37	**0.46**	0.41	**0.45**
AR(0.5) - S	0.02	**0.06**	0.11	**0.23**	0.15	**0.23**
AR(0.5) - M	0.14	**0.27**	0.41	**0.51**	0.47	**0.51**
AR(0.5) - L	**0.49**	0.47	0.60	**0.62**	0.60	**0.62**
AP(0.5:0.95) - (S,M,L)	0.14	**0.16**	0.23	**0.26**	**0.28**	**0.28**
AP(0.5:0.95) - S	0.01	**0.03**	0.07	**0.13**	0.09	**0.13**
AP(0.5:0.95) - M	0.07	**0.16**	0.25	**0.29**	0.30	**0.31**
AP(0.5:0.95) - L	**0.31**	0.29	**0.39**	0.38	**0.41**	0.39
AR(0.5:0.95) - S	0.01	**0.03**	0.07	**0.14**	0.10	**0.14**
AR(0.5:0.95) - M	0.08	**0.18**	0.29	**0.35**	0.34	**0.36**
AR(0.5:0.95) - L	**0.36**	0.35	**0.46**	**0.46**	**0.48**	0.47
Overall Score	4/15	11/15	3/15	13/15	3/15	12/15

vertex *x2,y2* by (3) with a group mean confidence c (4). The reason that this method is called GreedySlide is because of this final execution as it takes the biggest box consisting of every box in the group. By taking the biggest bounding boxes instead of the intersection, we enlarge the area of capturing an object rather than taking a bounding box that can be a partition of an object. Furthermore, to neglect tiny boxes that are bare to tell due to the zooming feature of this method, every box needs to be large than δ scale concerning the window.

$$x1 = \min_X G,\ y1 = \min_Y G,\ x2 = \max_X G,\ y2 = \max_Y G \quad (3)$$

$$c = \frac{1}{N}\sum_G c_g \quad (4)$$

IV. Experimental results

A. Benchmark Settings

Our goal is to provide an algorithm that can help to improve object detection model performance for quantized edge models and maintain a good generalization for those. Therefore, we addressed two experiments to verify the method's performance: external dataset validation and same dataset validation. We choose the Google Coral Board (which only supports int8 for inference), and Jetson Nano (for FP16 and FP32) as two most popular edge devices nowadays for our experiments. Noted that, in the different dataset scenarios, we emphasize the importance of our method at int8 settings.

Model and Dataset Selection: To test the detection performance, we address the GreedySlide detector on Google-provided models trained on the COCO dataset: SSD-MobilenetV2 (6M params) and Retina (32M params) model for object detections. We choose this model because it is one of the most famous architectures for edge object detection with fast and light capabilities. Moreover, it also emphasizes the flexibility of our approaches. Meanwhile, we use these three external datasets: KITTI, CrownAI and Autti, for a throughout benchmark when bringing the general detectors from COCO dataset to the narrow scope task like (vehicle detection/pedestrian detection). For KITTI dataset requirements, we follow their evaluation on three levels of difficulties: easy (big and clear), medium (average-size/slightly occlusion) and hard (small and highly occluded) with various of IoU rate. To apply that setting into CrownAI and Autti, we define the easy scope for bounding boxes that are bigger than 100 pixels and the hard scope for every scale of pixels.

Hyperparameters Selection: when evaluating on KITTI dataset, we have figured out: the ideal size k for sliding window is roughly 1/3 of the width of the image; γ confidence for each window should be 0.4 and γ' for the whole image is 0.7 to reach highest accuracy for detection. In addition, the reason we choose 0.7 is to balance between 0.6 and 0.8 result for generalization. Furthermore, this setting also allows our method to catch up to 70% speed of orignal models. Therefore, we use this configuration throughout our experiments. The benchmark of detection result is conducted and measured mAP in Pascal VOC Format.

B. Same Dataset Detection Performance

In this setting, we perform throughout evaluation on the pre-trained COCO dataset model published in an open repository of Google Coral and Jetson Nano. We omit the testing SSD-Mobinet v2 settings in Jetson Nano due to similar behavior results on Coralboard. We also discard the benchmark of the Retina model on Google Coral because of the slow inference of TPU on this device.

Overall the result at table I shows increased performance for the edge model regardless of int8 quantization, FP16 reduction, or full precision on 32 bits on both Jetson and Coralboard. However, it seems to lose a bit of performance for

TABLE II: COCO on traffic classes. N: non-GreedySlide. G: Greedyslide

Metrics \ Classes	Person		Car		Truck		Bus		Motorcycle		Bicycle	
	N	G	N	G	N	G	N	G	N	G	N	G
SSD-Mobilenet V2												
AP(0.5) - (S,M,L)	0.37	**0.43**	0.16	**0.29**	0.23	**0.27**	0.50	**0.57**	0.34	**0.4**	0.19	**0.30**
AP(0.5:0.95) - M	0.18	**0.28**	0.11	**0.28**	0.03	**0.13**	0.03	**0.21**	0.05	**0.14**	0.08	**0.19**
AP(0.5:0.95) - L	**0.53**	0.51	**0.49**	0.45	**0.40**	0.37	0.65	0.62	**0.47**	0.44	0.47	**0.49**
AR(0.5:0.95) - M	0.21	**0.31**	0.15	**0.34**	0.04	**0.17**	0.03	**0.23**	0.07	**0.18**	0.09	**0.22**
AR(0.5:0.95) - L	**0.60**	0.59	**0.57**	0.53	**0.46**	0.43	**0.69**	0.67	**0.53**	0.50	0.54	**0.56**
Overall Score	2/5	3/5	2/5	3/5	2/5	3/5	1/5	3/5	2/5	3/5	0/5	5/5
Retina												
AP(0.5) - (S,M,L)	**0.57**	**0.57**	0.43	**0.46**	0.31	**0.37**	0.64	**0.68**	0.52	**0.53**	0.38	**0.40**
AP(0.5:0.95) - S	0.15	**0.18**	0.17	**0.20**	0.05	**0.09**	0.14	**0.24**	0.11	**0.15**	0.06	**0.11**
AP(0.5:0.95) - M	**0.47**	0.46	**0.46**	0.45	0.16	**0.25**	0.37	**0.40**	**0.25**	0.24	**0.29**	0.28
AP(0.5:0.95) - L	**0.67**	0.61	**0.54**	0.48	**0.42**	0.37	**0.72**	0.68	**0.52**	0.51	**0.52**	0.51
AR(0.5:0.95) - S	0.17	**0.20**	0.18	**0.22**	0.06	**0.12**	0.15	**0.28**	0.11	**0.18**	0.06	**0.12**
AR(0.5:0.95) - M	**0.52**	0.50	**0.53**	**0.53**	0.21	**0.33**	0.39	**0.44**	**0.32**	0.29	**0.33**	**0.33**
AR(0.5:0.95) - L	**0.73**	0.69	**0.62**	0.58	**0.51**	0.50	**0.74**	0.73	**0.57**	0.55	**0.58**	**0.58**
Overall Score	5/7	3/7	4/7	4/7	2/7	4/7	2/7	5/7	4/7	3/7	4/7	5/7

TABLE III: mAP results for Greedy-SSD detector in comparison with SSD-MobilenetV2 and modified on KITTI, CrownAI and Autti Dataset

Models	easy (0.5)	medium (0.5)	hard (0.5)	easy (0.7)	medium (0.7)	hard (0.7)
KITTI						
Greedy-SSD	**0.5425**	**0.6421***	**0.5924**	**0.5162**	**0.5244**	**0.4204**
SSD-MobilenetV2	0.1632	0.1519	0.1517	0.1566	0.1176	0.1191
CrownAI						
Greedy-SSD	**0.6791***	**-**	**0.5672**	**0.5244**	**-**	**0.4019**
SSD-MobilenetV2	0.1877	-	0.1549	0.1172	-	0.0943
Autti						
Greedy-SSD	**0.6123***	**-**	**0.5772**	**0.5342**	**-**	**0.4446**
SSD-MobilenetV2	0.1784	-	0.1612	0.1123	-	0.1003

large objects but emphasizes the efficiency of small objects and medium-size ones. This can be understood as we fragment the picture to address, which makes the small objects more robust but accidentally divides the big objects into pieces. However, it is up to the shape and size of the objects in the picture. When inspecting some specific classes in Table II, we can notice there exists class can maintain good detection in large size like *bicycle* or *bus*, while others can degrade roughly. However, the results show promising in accuracy performance when gathering the good points among AP(average precision) and AR (average recall) that Greedyslide boosts the origin model (roughly triple the performance $12/15$ for overall classes and $5/7$ for specific classes inspection). In addition, it also shows the flexibility of our method on different model types (SSD and Retina) and edge configurations. It also hints our method can assist the model at full precision scale.

C. External Dataset Detection Performance

In this setting, we only address the SSD-Mobilenet as introduced in the Benchmark Settings. Table (III) summarizes the evaluation results from the three datasets. As shown, Greedy-SSD has overperformed to two baselines method for one-stage models. The accuracy mAP increases roughly five times in comparison with SSD-MobilenetV2, which are really sufficient detectors in computers or servers. For closer inspection, SSD-MobilenetV2, through quantization, has dramatically reduced its accuracy by six times according to SOTA full precision results on KITTI open dataset benchmark (from 0.61). Furthermore, Greedy-SSD performs better regardless of object scales in the image when the gap between easy and hard task is not severe. One of the most important factors for this method is that it can help the SSD-MobilenetV2 increase its accuracy to acceptable results without designing a new deep learning model structure for quantization like other approaches.

V. Conclusion & Future Work

In this paper, we have introduced our novel GreedySlide method on optimizing detection performance on the different type of datasets, especially for edge devices' scopes. It proves that the gap of detection performance with and without GreedySlide is considerable and well-improved in external dataset. Using GreedySlide can reduce the complexity and effort to optimize a model by quantization and ensure a better generalization for the model. In the future, we will try to integrate this behavior into the training pipeline as a multi-scale feature model for a better and more compact solution.

VI. Acknowledgement

This research is funded by University of Transport Technology (YTT) under grant number ĐTTĐ2021-06

REFERENCES

[1] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[2] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4852–4861, 2019.

[3] Samuel Greengard. Ai on edge. *Commun. ACM*, 63(9):18–20, August 2020.

[4] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[5] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[6] Jangho Kim, KiYoon Yoo, and Nojun Kwak. Position-based scaled gradient for model quantization and pruning. 2020.

[7] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[8] Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, Yongkweon Jeon, Baeseong Park, and Jeongin Yun. Flexor: Trainable fractional quantization. *arXiv preprint arXiv:2009.04126*, 2020.

[9] Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. Fully quantized network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2810–2819, 2019.

[10] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[11] W Liu, D Anguelov, D Erhan, C Szegedy, S Reed, CY Fu, and AC Berg. Ssd: Single shot multibox detector. arxiv 2016. *arXiv preprint arXiv:1512.02325*, 2020.

[12] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[13] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. arxiv 2018. *arXiv preprint arXiv:1804.02767*, pages 1–6, 2018.

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[16] S. Zhai, D. Shang, S. Wang, and S. Dong. Df-ssd: An improved ssd object detection algorithm based on densenet and feature fusion. *IEEE Access*, 8:24344–24357, 2020.

[17] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.