

# Anchor pruning for object detection

Maxim Bonnaerens\*, Matthias Freiberger, Joni Dambre

IDLab-AIRO, Ghent University - imec, Gent, Oost Vlaanderen, Belgium

## ARTICLE INFO

Communicated by Nikos Paragios

MSC:  
68T07  
68T45

Keywords:  
Object detection  
Pruning  
Real time

## ABSTRACT

This paper proposes anchor pruning for object detection in one-stage anchor-based detectors. While pruning techniques are widely used to reduce the computational cost of convolutional neural networks, they tend to focus on optimizing the backbone networks where often most computations are. In this work we demonstrate an additional pruning technique, specifically for object detection: anchor pruning. With more efficient backbone networks and a growing trend of deploying object detectors on embedded systems where post-processing steps such as non-maximum suppression can be a bottleneck, the impact of the anchors used in the detection head is becoming increasingly more important. In this work, we show that many anchors in the object detection head can be removed without any loss in accuracy. With additional retraining, anchor pruning can even lead to improved accuracy. Extensive experiments on SSD and MS COCO show that the detection head can be made up to 44% more efficient while simultaneously increasing accuracy. Further experiments on RetinaNet and PASCAL VOC show the general effectiveness of our approach. We also introduce ‘overanchored’ models that can be used together with anchor pruning to eliminate hyperparameters related to the initial shape of anchors. Code and models are available at [https://github.com/Mxbonn/anchor\\_pruning](https://github.com/Mxbonn/anchor_pruning).

Object detection is one of the most widely studied tasks in computer vision. It requires not only predicting the class but also the exact location of each object in an image. Object detection models are mainly categorized into two categories based on whether detection happens in one or two stages. Two-stage networks (Cai and Vasconcelos, 2018; Girshick, 2015; Ren et al., 2015), first run a regional proposal network (RPN) that produces class-agnostic candidate object locations. These proposals are then used as input for a second stage where the proposed object regions are classified and the location of the object is refined. One-stage networks (Lin et al., 2017b; Liu et al., 2016; Redmon and Farhadi, 2017; Sermanet et al., 2013; Tan et al., 2020) use a single convolutional neural network with a predefined set of default anchors (i.e. an anchor associates a predefined size and aspect ratio to each pixel at a feature map) of which the classes and location offsets are directly predicted. While two-stage networks are generally the most accurate, one-stage detectors tend to be more resource-efficient and are often the only choice to run in real-time on resource-constrained devices, such as mobile platforms (Huang et al., 2017).

Many studies aimed at creating more efficient models for object detection have been carried out in recent years. Most have achieved it through slimmer backbone networks (e.g. MobileNet in SSDLite Sandler et al., 2018) or compression techniques such as weight pruning and quantization (Deng et al., 2020; Liu et al., 2018). And while there has been some work on anchor-free detectors (Law and Deng, 2018; Tian et al., 2019), most mainstream single-shot detectors still use the

same anchor-based approach (Lin et al., 2017b; Redmon and Farhadi, 2018; Tan et al., 2020). As computational complexity in the backbone network keeps shrinking, the importance of an efficient object detection head, in which anchors are used to predict potential objects, increases. The influence of these anchors on the computational cost of the object detector is twofold. Firstly, the number of anchors an object detector uses directly influences the size of the convolutional layers in the head of the detection model. Secondly, all bounding boxes produced by those anchors need to go through a non-maximum suppression (NMS) post-processing step to determine the final detected objects. Previous studies on the speed/accuracy trade-off for object detectors have shown that for small models, i.e. the ones typically used in an embedded context, “NMS can take up the bulk of the running time” (Huang et al., 2017). While the exact running time of the post processing step depends strongly on the used hardware, systematic assessments (Verucchi et al., 2020; Cai et al., 2019) have shown that the number of detections coming out of an object detection network has a non-negligible influence on the end-to-end inference time. Whereas traditional backbone pruning techniques only influence part of the model inference time, anchor pruning can reduce the running time of both the model inference and the post-processing step.

In this paper we introduce a novel anchor pruning technique and evaluate its effect on the accuracy and efficiency of one-stage anchor-based object detection models. The main contributions of this work include:

\* Corresponding author.

E-mail address: [maxim.bonnaerens@ugent.be](mailto:maxim.bonnaerens@ugent.be) (M. Bonnaerens).

(1) We propose an anchor pruning search algorithm that determines which key anchors to remove and identifies several anchor configurations on the accuracy/resources Pareto front, allowing model developers to trade accuracy for performance when resources are constrained.

(2) We show that the accuracy of the proposed pruned models can be further improved through fine-tuning or retraining from scratch. By removing well-chosen anchors and retraining, it is possible to achieve both better efficiency and accuracy compared to the unpruned baseline model.

(3) We introduce the concept of an ‘overanchorized’ model. By starting our proposed anchor pruning approach on an object detector that has many closely overlapping anchors, we show that we can avoid tuning hyperparameters related to the initial shape of the anchors.

(4) We extensively benchmark our approach on the SSD and RetinaNet object detectors on both the MS COCO and PASCAL VOC datasets. The results strongly demonstrate that pruning anchors and retraining can reduce the resource costs of an object detector significantly and that our method generalizes to different anchor-based single shot detectors.

## 1. Related work

### 1.1. Model scaling

Many real-world object detection applications run on systems where model size and inference speed are highly constrained. The object detection networks used in such systems are not selected based on absolute state-of-the-art accuracy but on the best possible performance given these constraints. To this end, many model scaling techniques are applied to scale a model towards a certain size or speed, where the size is determined by the number of parameters in the network and the speed by the number of floating point operations. In object detection, it is common to downscale the model by changing the backbone network (e.g. from VGG [Simonyan and Zisserman, 2014](#) to MobileNet [Sandler et al., 2018](#)), by reducing the input resolution (e.g. from  $512 \times 512$  to  $300 \times 300$  in SSD [Liu et al., 2016](#)), by reducing the number of layers (e.g. from 50 to 18 in ResNet [He et al., 2016](#)) or a combination of these ([Tan et al., 2020](#)). To further trim the computational cost in those models, techniques such as pruning ([Liu et al., 2018](#)) and quantization ([Zhou et al., 2017](#)) are often used, possibly in combination with knowledge distillation ([Hinton et al., 2015](#)) to recover any lost accuracy. Earlier pruning techniques focused on removing individual weights or connections, but in recent years, the trend has shifted towards more structural pruning at the level of filters or even entire layers. Compared to pruning many individual weights, structural pruning does not need specialized hardware or libraries to benefit from the sparsity.

However, none of the above scaling techniques are specific to object detection. We will show in Section 2 that our proposed method of pruning anchors is an effective way to downscale the computational complexity of the object detection head. It is orthogonal to the previously mentioned scaling methods, meaning that they can be combined to achieve better compression results.

### 1.2. Anchor-based

Anchor-based one-stage detectors are convolutional object detection models that consist in most cases of three parts. The *backbone* network is an off-the-shelf convolutional network, trained originally for image classification, that extracts features for detecting objects. On top of this backbone, most object detection models have a middle part, the *neck*, that adds more layers after the backbone to produce better adapted features. In SSD ([Liu et al., 2016](#)) the neck consists of additional convolutional feature layers, while in more recent networks such as RetinaNet ([Lin et al., 2017b](#)) and EfficientDet ([Tan et al., 2020](#)) some form of Feature Pyramid Network (FPN) ([Lin et al., 2017a](#)) is used,

that has additional lateral connections with the backbone. The final part is the *object detection head* which includes a classifier and regressor to predict classes and exact locations of the objects in the image. The detection head is applied to several feature layers which can come from both the backbone and the neck.

Anchor-based detectors associate some predefined *anchors*, sometimes also named default bounding boxes or priors, to each feature layer to which the detection head is attached. These anchor-associated feature layers are what we will from now on refer to as feature maps. Usually, the anchors are defined in terms of size and aspect ratio. The classifier and regressor in the detection head output the class scores and the 4 offsets relative to the predefined anchor shape for each pixel in a feature map. While many papers claim that an increase in the number of anchors leads to an increase in accuracy ([Li et al., 2019](#); [Liu et al., 2016](#)), we will show in Section 3 that this does not always hold and that too many anchors can lead to a decrease in accuracy.

### 1.3. Anchor-free

Some one-stage detectors use a different approach. The first version of YOLO predicted the coordinates of bounding boxes directly using fully connected layers ([Redmon et al., 2016](#)), the latest versions have however also switched to using anchors. More recently proposed anchor-free detectors have deliberately moved away from using anchors. CornerNet ([Law and Deng, 2018](#)) for example, detects corners of an object and groups multiple predictions together to produce a final bounding box. One of the reasons of existence for FCOS ([Tian et al., 2019](#)), another anchor-free object detector, is that anchor shapes need to be carefully tuned. While anchor-free methods are growing in popularity and show promising results, their alternatives to anchors often mean that the detections made in the object detection head are fixed and cannot be downsampled to trade accuracy for performance. We believe that anchor-based solutions are still relevant as they are widely used and as demonstrated in EfficientDet ([Tan et al., 2020](#)), are able to achieve state-of-the-art results in both accuracy and efficiency.

### 1.4. Anchor shape optimization

One of the reasons to use an anchor-free approach is that the number of anchors and their predefined shapes can be seen as hyperparameters that need to be carefully tuned. As stated earlier, anchors are usually defined in terms of size (or scale) and aspect ratio. The optimal aspect ratios depend strongly on the domain in which the object detector is applied. For face recognition square aspect ratios will be important ([Zhang et al., 2017](#)) while for text detection such as in license plate recognition wider aspect ratios like 1 : 5 need to be included ([Liao et al., 2018](#)).

The optimal scales for the anchors in each layer, not only depend on the domain (i.e. the distribution of the size of the objects) but also on the receptive field and the stride of the associated feature map. The theoretical receptive field indicates which pixels in the input image affect the value of a pixel in the feature map. However as shown in [Luo et al. \(2016\)](#), the effective input region that has a non-negligible impact on this feature map value is only a fraction of the theoretical receptive field. The stride of the feature map determines the interval at which anchors are placed. For example, in SSD the stride size of the first anchor-associated feature map is 8 pixels, indicating that the anchors produce a bounding box every 8 pixels on the input image.

To reduce the importance of the initial anchor shapes, RefineDet ([Zhang et al., 2018](#)) proposes a one-stage object detector with an additional module to refine anchor locations, alleviating the negative effects of suboptimal anchor shapes. Other approaches such as Guided Anchoring ([Wang et al., 2019](#)) and MetaAnchor ([Yang et al., 2018](#)) remove the need for hand-picked anchors all together by including additional components that generate the anchors. However, this requires extra resources and results in a larger model and longer inference time.

More recent approaches dynamically learn the anchors during training, which increases training time but has no influence on the model size and inference time. FreeAnchor (Zhang et al., 2019) and MAL (Ke et al., 2020) start from a bag of candidate anchors and learn which ones are optimal, Anchor Box Optimization (Zhong et al., 2020) adapts the loss and changes the anchor shapes during back-propagation. These approaches optimize the anchor shape initialization but still leave the number of anchors as a hyperparameter. While these approaches benefit from anchor selection during training, this also means that any modification to the numbers of anchors requires retraining the network, making it infeasible to use such methods to efficiently explore accuracy/performance trade-offs. In comparison our proposed pruning method can obtain an accuracy without any need for re-training or fine-tuning. Only when a pruned configuration is selected does our method require fine-tuning or retraining.

Our anchor pruning approach is compatible with these previous anchor shape optimization methods as it does not change the shapes of the initial anchors but rather removes the redundant or least important anchors. Models that have optimized anchor shapes can be used as a baseline from which to start anchor pruning.

## 2. Anchor pruning

In this section, we propose an efficient way to explore different anchor configurations through pruning. We first demonstrate why pruning anchors is a sensible strategy. Next, we present a search algorithm to determine which anchors to remove for different resource constraints. The result is a sequence of anchor configurations that form a Pareto front in the accuracy/performance design space. This allows model designers to trade-off between accuracy and performance when selecting their final model.

### 2.1. Redundant anchors

In a one-stage anchor-based object detector, the total number of predicted bounding boxes per image is  $N = \sum_{i=1}^k N_i = \sum_{i=1}^k (A_i \times H_i \times W_i)$ , where  $H_i$  and  $W_i$  are the size of the feature maps, and  $A_i$  is the number of anchors, for  $k$  different feature map layers. In some object detection architectures  $N$  also depends on the number of classes  $C$  but since most recent models separate class-agnostic bounding box regression from the object classification, we do not include it here. Our proposed technique can be trivially extended to handle class-specific anchors.

When  $\forall i, j \in 1, \dots, k : A_i = A_j$  and  $H_{i+1} = H_i/2$ ,  $W_{i+1} = W_i/2$ , as is common in single stage detectors where the feature maps are often in a pyramid structure, then  $N_1 \approx \frac{3N}{4}$ . This means that the majority of the predicted bounding boxes come from the anchors associated with the first feature map.

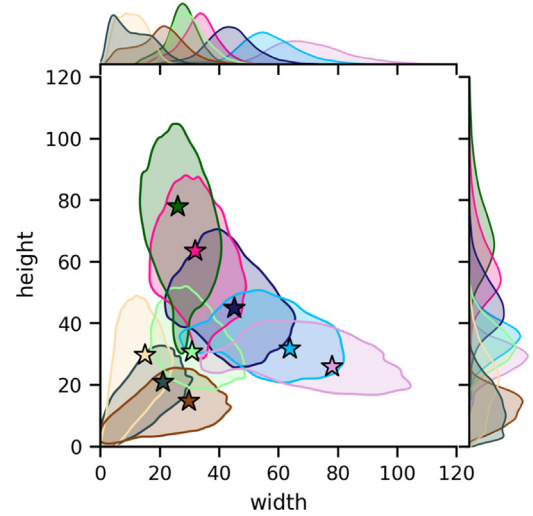
Table 1 shows the relation between the number of anchors per feature map layer and the total number of bounding boxes as well as the accuracy on the COCO dataset for different SSD300 models. The first row shows the results when using the 6 anchors per layer as defined in the scales and aspect ratios section of the original SSD paper (Liu et al., 2016). The second row shows the results for the final SSD300 version reported in the experimental results section of the original paper where the number of anchors is reduced from 6 to 4 in the first and last two layers. While the original paper does not motivate why some layers have only 4 instead of 6 anchors, it can be seen from the results in the table that the choice likely comes from the observation that 6 anchors in the first layer offer less accuracy at a higher computational cost. The last row in Table 1 shows that further reducing the number of anchors in that first layer could have improved accuracy and efficiency even more. As we argued earlier, the optimal number of anchors is a hyperparameter that requires careful tuning. This example also indicates that more anchors do not always mean better accuracy, as will be further shown in Section 3.

**Table 1**

Relation between the number of anchors per layer  $A_i$  and the total number of bounding boxes  $N$  in SSD300. The number of anchors  $A_1$  in the first feature map layer have the largest influence on  $N$ . It can be seen that reducing this number has a positive influence on both the mean average precision (mAP) and the number of bounding boxes.

| $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ | $N$    | COCO mAP |
|------------------------------------|--------|----------|
| {6, 6, 6, 6, 6, 6}                 | 11 640 | 25.6     |
| {4, 6, 6, 6, 4, 4} <sup>a</sup>    | 8732   | 25.7     |
| {2, 6, 6, 6, 4, 4}                 | 5844   | 25.8     |

<sup>a</sup>Marks the original configuration of SSD300 (Liu et al., 2016).



**Fig. 1.** The distribution of the shape of bounding boxes for each anchor in the first two layers of SSD300 on MS COCO. The markers correspond to the default shape of each anchor, and the matching colored region to the final bounding boxes produced by that anchor. Plotted on the side are the marginal distributions of the width and height.

Analysis of the bounding boxes predicted by each anchor, shows that different anchors can predict similar bounding box shapes. For example, an object with a ground truth bounding box of aspect ratio 1 : 1.5 could be predicted by a 1 : 1 anchor and a 1 : 2 anchor. Fig. 1, shows the distribution of the predicted bounding boxes for each anchor in the first two layers of SSD300 on the MS COCO dataset. It can be observed that certain anchors produce bounding boxes that are almost completely covered by predictions of neighboring anchors, which intuitively explains why it is possible to drop certain anchors without loss of performance.

### 2.2. Optimal anchor search

Unlike methods such as magnitude-based pruning of convolutional filters (Li et al., 2016), we not only have to decide how much to prune but also what to prune. Anchors that produce bounding boxes that are largely covered by neighboring anchors are more optimal to prune than those producing unique shapes. The number of possible anchor subsets that result from pruning a model is equal to  $|\mathcal{P}(\mathcal{A})| = 2^{|\mathcal{A}|}$  where  $\mathcal{A}$  is the set of all anchors in the unpruned model and  $\mathcal{P}(\mathcal{A})$  stands for the power set of  $\mathcal{A}$ . Models such as SSD and RetinaNet have 30 and 45 anchors respectively, which makes it infeasible to evaluate all combinations. We therefore introduce a greedy search algorithm to efficiently explore the search space. This procedure is summarized in Algorithm 1.

Our anchor pruning algorithm starts from a fully (pre-)trained model  $M$  with anchor configuration  $C = \mathcal{A}$ . The algorithm iterates over

**Algorithm 1** Anchor Pruning Search

---

**Input:** Fully trained model  $M$  with anchors  $\mathcal{A}$   
**Output:** Pareto Frontier  $P$

```

1: Set  $S \leftarrow \{\mathcal{A}\}$  and  $P \leftarrow \{\mathcal{A}\}$ 
2: while  $S \neq \emptyset$  do
3:   Select a configuration  $C$  from  $S$  to explore
4:   for each anchor  $a_i \in C$  do
5:      $C_i = C \setminus a_i$ 
6:      $\text{accuracy}_{C_i}$  = Accuracy of  $M$ , keeping only
       predictions produced by anchors in  $C_i$ 
7:      $\text{cost}_{C_i}$  = Resource cost of  $M$  with anchors in  $C_i$ 
8:     Compare  $\text{accuracy}_{C_i}$  and  $\text{cost}_{C_i}$  to
        $\text{accuracy}_{C_j}$  and  $\text{cost}_{C_j}$  from all  $C_j \in P$ 
9:     if  $C_i$  is Pareto Optimal in  $P$  then
10:      Add  $C_i$  to  $P$  and  $S$ 
11:     end if
12:   Remove any  $C_i$  that is no longer optimal from  $P$ 
13:   end for
14: end while

```

---

an adapting set of unexplored anchor configurations  $S$  and constructs a set of Pareto-efficient anchor configurations  $P$ . Starting from  $S = \{\mathcal{A}\}$  and  $P = \{\mathcal{A}\}$ , we take an unexplored configuration  $C$  ( $C \subseteq \mathcal{A}$ ) from  $S$  at each iteration until  $S = \emptyset$ . From anchor configuration  $C$  we evaluate configurations  $C_1, \dots, C_n$ , where  $C_i = C \setminus a_i$ , with anchor  $a_i \in C$ .

Evaluating a configuration  $C$  means evaluating the accuracy of the model  $M$  when only using the predictions made by anchors  $a_i \in C$ . This can be done in an efficient way by initially storing all predicted bounding boxes from  $M$ , i.e. before any are discarded by post-processing steps such as NMS, and to re-evaluate by only keeping the bounding boxes produced by anchors  $a_i$  in the configuration  $C$ . This avoids the costly process of running all validation inputs through an adapted model for each new configuration. The evaluation of the accuracy metric should be done on a validation set to avoid overfitting the anchor configurations on the test set. As stated before, we are not just interested in the highest accuracy but in the accuracy/resources trade-off. This requires evaluating an accuracy metric (e.g. mean average precision (mAP), F-score, etc.) and a resource metric (e.g. inference speed, FLOPs, model size, number of predicted bounding boxes, etc.). Resource metrics that can be calculated such as FLOPs should be preferred above metrics that need to be experimentally measured such as inference speed.

A configuration is Pareto-optimal or Pareto-efficient when no other node that uses fewer or equal resources achieves higher accuracy. At any point during the search phase,  $P$  stores the Pareto-optimal configurations that have been evaluated thus far. To achieve this, the accuracy and resource cost of  $C_i$  are compared to all  $C_j \in P$ , if  $C_i$  is Pareto optimal in  $P$  we add  $C_i$  to  $P$  and  $S$ , and any  $C_j$  that is no longer Pareto optimal is removed from  $P$ . To speedup the running time of the search algorithm, one could also add an accuracy threshold  $\theta$  such that  $C_i$  is only added to  $P$  and  $S$  if it is Pareto Optimal in  $P$  and  $\text{accuracy}_{C_i} \geq \theta$ . This is often useful in practice when we are only interested in accuracy/resources trade-offs as long as the accuracy remains acceptable. Once  $S$  is empty,  $P$  contains all Pareto-efficient anchor configurations.

Because the search is greedy,  $P$  might contain configurations that are suboptimal solutions as the algorithm may miss anchor combinations that would arise from further pruning discarded configurations. However, as we will show in Section 3, our method significantly outperforms random pruning and the default pruning approach from RetinaNet.

**Table 2**

The number of FLOPs for different one-stage object detectors, along with the percentage of the FLOPs that take place in the head of the network. FLOPs are calculated for an input resolution of  $300 \times 300$ .

| Model                     | FLOPs | FLOPs(%) head |
|---------------------------|-------|---------------|
| SSD                       | 34.4B | 12.3%         |
| RetinaNet                 | 21.7B | 57.6%         |
| MobileNetV3-Small-SSDLite | 0.2B  | 37.5%         |

### 3. Experiments

#### 3.1. Setup

Our proposed anchor pruning is rather general and can be applied to any anchor-based one-stage object detector. We demonstrate our results primarily on SSD300 (Liu et al., 2016), one of the most used and influential one-stage object detectors. SSD uses a VGG16 backbone, a pyramid of convolutional feature maps in the neck, and a head with a  $3 \times 3 \times (A_i \times (\text{Classes} + 4))$  convolution on top of each of the 6 feature maps. Each feature map has anchors of a fixed scale and with aspect ratios of  $\{1, 2, \frac{1}{2}, 3, \frac{1}{3}\}$  and an additional anchor with aspect ratio 1 but a larger scale (which we will refer to as  $1^+$ ), resulting in 6 anchors per layer. However for the first and the two last feature maps, the anchors with aspect ratios 3 and  $\frac{1}{3}$  are removed, resulting in a total of 30 anchors.

While SSD is no longer a state-of-the-art object detection model, as many alternatives that are both more accurate and more efficient have been developed since its publication, the fundamental structure of the Single-Shot-Detector remains used in most recent object detectors.

Table 2 shows the FLOPs (multiply-adds) of different object detection models along with how many of those FLOPs are in the detection head. It can be seen that the relative importance of the head has increased as backbones got more efficient in more recent object detectors. Given the growing focus on resource-efficient models, we expect this trend to continue. We will show in Section 3.6 that our method generalizes well to these more recent anchor-based one-staged object detectors.

As stated in the introduction, the running time of an object detection model in an embedded context is often dominated by the running time of the post processing steps, which is directly related to the number of bounding boxes produced by the network. In systems where this post-processing bottleneck is not present, the running time will be directly related to the FLOPs of the model, and as our approach is backbone independent we only report on those FLOPs in the head. In the remainder of this work we report the resource cost as FLOPs in the head of the network or as the number of bounding boxes it produces. Compared to reporting latency which is very implementation and hardware dependent, we believe that our reported metrics allow model designers to better estimate the accuracy/resource trade-offs that are possible for their platform.

We evaluate pruning anchors of the SSD model on the MS COCO 2017 detection dataset (Lin et al., 2014). Before pruning, the SSD model is trained on train2017 using SGD for 120 epochs with a learning rate of  $10^{-3}$ , which is decreased to  $10^{-4}$  on epoch 80 and to  $10^{-5}$  on epoch 110, with a weight decay of  $5 \times 10^{-4}$  and a momentum of 0.9. The input images are resized to  $300 \times 300$  and all data augmentations as described in Liu et al. (2016) are used. The accuracy metric used for COCO is the mean average precision (mAP) evaluated at intersection-over-union (IoU) thresholds evenly distributed between 0.5 and 0.95, for the resource metric we use the number of FLOPs in the object detection head. Our anchor pruning search, uses val2017 to evaluate candidate anchor configurations.



**Table 3**

Results on COCO test-dev2017 from the evaluation server for different anchor configurations of SSD. The models are ordered in decreasing number of bounding boxes and FLOPs. AP<sub>S,95</sub> is the COCO mAP, AR the average precision and subscripts S/M/L correspond to small, medium and large objects. The superscripts<sup>2,3,4,5</sup> refer to the subsection of Section 3 where the details of the relevant experiments can be found.

| Model (SSD variant)                                      | AP <sub>S,95</sub> | AP <sub>S0</sub> | AP <sub>75</sub> | AP <sub>S</sub> | AP <sub>M</sub> | AP <sub>L</sub> | AR <sub>S</sub> | AR <sub>M</sub> | AR <sub>L</sub> | FLOPs head | BBoxes |
|----------------------------------------------------------|--------------------|------------------|------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|------------|--------|
| SSD baseline                                             | 25.7               | 44.0             | 26.6             | 7.1             | 27.1            | 41.6            | 11.2            | 39.9            | 57.6            | 4231M      | 8732   |
| SSD {1, 2, $\frac{1}{2}$ , 1 <sup>+</sup> } <sup>4</sup> | 25.5               | 43.7             | 26.2             | 7.3             | 26.6            | 40.8            | 11.6            | 39.6            | 56.5            | 3577M      | 7760   |
| SSD {1, 1 <sup>+</sup> } <sup>4,5</sup>                  | 25.0               | 43.3             | 25.5             | 7.2             | 25.6            | 39.6            | 12.7            | 39.0            | 56.1            | 1788M      | 3880   |
| Configuration-A <i>pruned</i> <sup>2,3</sup>             | 25.6               | 43.9             | 26.5             | 7.1             | 27.1            | 41.2            | 11.2            | 39.8            | 57.2            | 3607M      | 7814   |
| Configuration-A <i>fine-tuned</i> <sup>3</sup>           | 25.7               | 44.0             | 26.6             | 7.2             | 27.2            | 41.3            | 11.3            | 39.7            | 57.3            | 3607M      | 7814   |
| Configuration-A <i>retrained</i> <sup>3</sup>            | 25.5               | 44.0             | 26.3             | 7.6             | 27.2            | 40.4            | 11.8            | 39.8            | 56.4            | 3607M      | 7814   |
| Configuration-B <i>retrained</i> <sup>3,4</sup>          | 25.8               | 44.5             | 26.5             | 6.8             | 27.2            | 41.2            | 12.0            | 40.1            | 57.3            | 2476M      | 4926   |
| Configuration-C <i>retrained</i> <sup>3,4,5</sup>        | 25.4               | 44.3             | 25.7             | 6.5             | 25.7            | 41.6            | 12.5            | 38.2            | 58.0            | 1628M      | 3121   |
| Configuration-D <i>retrained</i> <sup>3</sup>            | 23.1               | 41.5             | 23.1             | 3.7             | 22.6            | 41.4            | 7.3             | 35.4            | 58.0            | 774M       | 1291   |
| Pruned Layer-wise <sup>4</sup>                           | 25.0               | 43.6             | 25.5             | 7.5             | 26.5            | 38.1            | 13.0            | 39.3            | 54.8            | 1788M      | 3880   |
| Overanchorized <sup>5</sup>                              | 25.8               | 43.5             | 27.0             | 6.6             | 28.4            | 42.1            | 9.7             | 40.7            | 59.1            | 6673M      | 13584  |
| Pruned Overanchorized <sup>5</sup>                       | 25.3               | 44.0             | 25.7             | 6.5             | 25.6            | 42.1            | 12.3            | 37.4            | 58.8            | 1620M      | 3080   |

### 3.2. Pruning

Fig. 2 shows the Pareto frontier of pruned anchor configurations as obtained by our search algorithm, representing points at which fewer FLOPs can only be achieved by sacrificing accuracy. We also highlight the original unpruned model and plot the results of configurations achieved when pruning anchors randomly. As can be seen in the figure, our method outperforms random pruning for all FLOPs, indicating our effectiveness in pruning anchors that are redundant or that lead to minimal accuracy loss. For pruned models that have a large reduction in FLOPs and therefore require more anchors to be pruned, selecting the right anchors can be the difference between a model with acceptable accuracy and a model that is no longer usable because the accuracy dropped too much. Note that the wider gaps on the  $x$ -axis between anchor configurations in the Pareto frontier are due to pruned anchors that have a large influence on the number of FLOPs, such as anchors in the first layer of the head.

The most accurate configuration produced by our search algorithm, which we will refer to as Configuration-A *pruned*, is able to reduce the number of FLOPs in the detection head by 15% without losing any accuracy on the val2017 dataset. Table 3 shows the accuracy in more detail on the test-dev2017 dataset, indicating that the accuracy degraded slightly for large objects. This is not surprising as Configuration-A pruned 5 of the 8 largest anchors.

### 3.3. Fine-tuning versus retraining

Most pruning techniques fine-tune the model after pruning to recover part of the lost accuracy. In this section, we show the impact of fine-tuning after pruning anchors and compare it to retraining the model from scratch with the new anchor configuration. When fine-tuning we train for 10 additional epochs with a learning rate of  $10^{-5}$ , for retraining we use the same settings as described in Section 3.1. Fig. 3 shows the effect of fine-tuning and retraining on four highlighted configurations. As illustrated in the figure, fine-tuning does recover some of the lost performance during pruning. However, for all nodes except Configuration-A, retraining the configuration from scratch achieves much higher accuracy. As more anchors get pruned, such as in Configuration-D, the difference between fine-tuning and retraining increases. The Pareto frontier produced by Algorithm 1 can be seen as a lower bound for the accuracy that can be achieved for those configurations after fine-tuning or retraining.

On the val2017 dataset there are now multiple configurations that match the unpruned baseline accuracy: Configuration-A *pruned*, Configuration-A *fine-tuned* and Configuration-B *retrained*. Further evaluation on the test-dev2017 dataset, as shown in Table 3, indicates that Configuration-A *fine-tuned* is able to restore the lost performance as reported in the previous section and Configuration-B *retrained* is able to reduce the FLOPs and the number of bounding boxes by around 43% while even slightly improving accuracy.

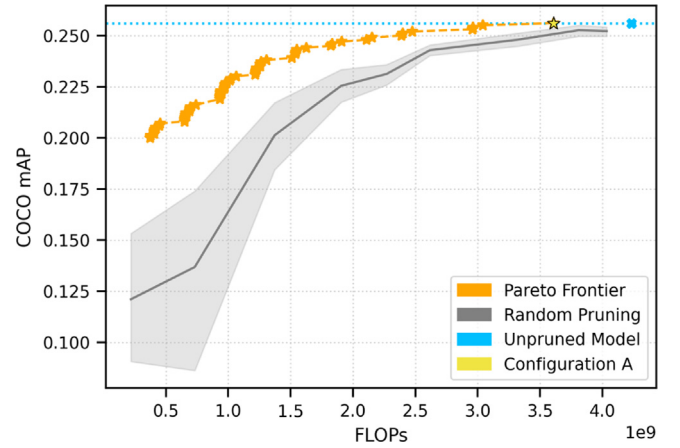


Fig. 2. Accuracy and FLOPs trade-offs for different optimal anchor configurations found by our anchor pruning search. The Pareto frontier is highlighted in orange, the unpruned baseline in blue. Accuracy is measured on the COCO val2017 dataset and FLOPs indicate the multiply-adds in the detection head. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

### 3.4. Layer-wise pruning

Experimenting with anchor configuration in the head of an object detector is not new, but it is normally done before training and in a symmetric way. Adding or removing anchor shapes is done across all layers and when the aspect ratio  $x$  is added/removed,  $\frac{1}{x}$  is also added/removed. The original SSD paper does mention two additional anchor settings compared to the baseline; one where aspect ratios 1 and  $\frac{1}{3}$  are removed, and one which is further reduced by removing aspect ratios  $2, \frac{1}{2}$ . Just like our retraining approach, these additional models are trained from scratch rather than created by pruning the larger baseline. We compare these ‘default’ settings to configurations in our Pareto frontier that have a comparable number of FLOPs. The SSD setting where aspect ratios  $3, \frac{1}{3}$  are removed has a performance drop of 0.2% on COCO. In the previous subsection, we already showed that we can prune much more than that while simultaneously improving accuracy. The SSD setting with only two square anchors per layer has a performance drop of 0.7%. Configuration-C *retrained* has a comparable number of FLOPs to this setting but only results in 0.3% reduced accuracy. We also compare to a pruned model where the anchors are pruned layer by layer until 2 anchors per layer remain. While the layer-wise pruning configuration keeps other aspect ratios besides 1 : 1, it does result in comparable accuracy (after retraining) to the last mentioned SSD setting, indicating the importance of pruning freely over

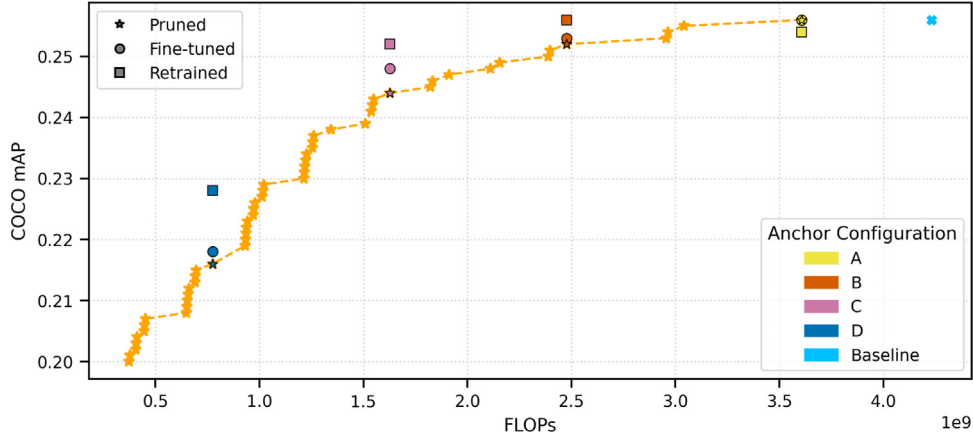


Fig. 3. Accuracy of fine-tuning and retraining certain highlighted configurations for SSD on COCO val2017. Fine-tuning improves accuracy after pruning but in most cases, retraining achieves a much higher accuracy.

all layers as we do in Algorithm 1. The detailed results can be found in Table 3.

### 3.5. Overanchored model

As explained in Section 1, predefined anchors introduce many additional hyperparameters that require careful tuning. We introduce an ‘overanchored’ model and define it as an object detection model that has more anchors than strictly needed. For applications where computational resources are not constrained, our previous experiments showed that while more anchors do not always improve accuracy, they also do not degrade the accuracy significantly. For applications with constrained-resources we can prune from an overanchored model to arrive at an optimized anchor configuration. This also allows our algorithm to find a suitable set of anchors if no initial anchors with good performance are known. As an experiment, we defined an overanchored SSD model with 48 anchors (see Fig. 4) and pruned it to a model where the number FLOPs is below or equal to the SSD setting with two anchors per layer. Fig. 4 plots the anchors selected by the pruning step. The detailed results in Table 3 show that pruning an overanchored model achieves comparable accuracy to pruning the original baseline. This suggests that training an overanchored object detection model followed by anchor pruning can eliminate extensive tuning of the anchor shapes, while simultaneously removing the number of anchors as a hyperparameter.

### 3.6. RetinaNet

As stated earlier, our approach is general as it can be applied to different one-stage anchor-based object detectors. In this subsection we demonstrate that our technique is also successful on more recent object detectors such as RetinaNet. The default configuration of RetinaNet uses 9 anchors in each layer by combining 3 scales and 3 aspect ratios  $\{\frac{1}{2}, 1, 2\}$ . As shown earlier in Table 2, unlike most object detectors, the majority of FLOPs in this model take place in the head. The explanation for this is that whereas the head in SSD directly does a  $3 \times 3 \times (A_i \times (Classes + 4))$  convolution, RetinaNet first applies 4 additional  $3 \times 3 \times 256$  convolutional layers. Another difference with SSD is that these prediction layers in the detection head are reused on all feature maps. To account for these differences, we apply the following changes to Algorithm 1: Configuration C generates  $C_i$  by either removing a certain anchor configuration on each layer or by removing all anchors of a certain layer (as the prediction layers are shared, this practically means not applying the head to a certain feature map).

An important side effect of the additional convolutional layers in the RetinaNet head, is that the decision not to apply the head to a certain feature map has a large influence on the number of FLOPs but not

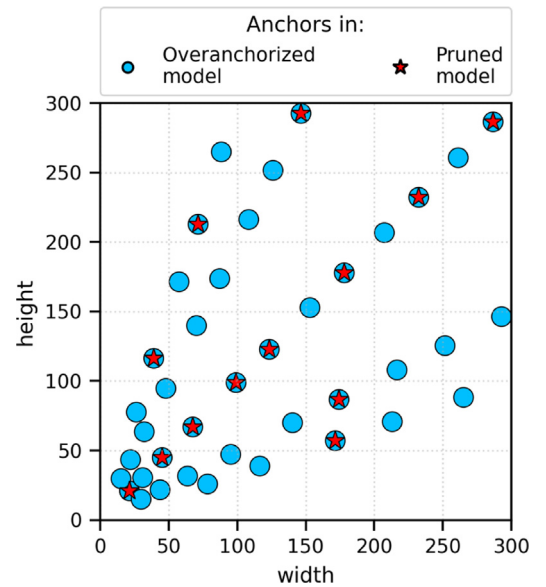


Fig. 4. Anchor shapes for the ‘overanchored’ SSD model and the anchors that remain after pruning. The overanchored model has 48 anchors that produce 13,584 bounding boxes and achieves 25.8 mAP on COCO. The pruned version only has 14 anchors producing 3080 bounding boxes and achieves 25.4 mAP.

necessarily on the number of bounding boxes. Because of this property, we use RetinaNet to show the difference between optimizing for the number of FLOPs or optimizing the number of bounding boxes. When optimizing an object detector for a certain use-case, the used hardware will determine whether the computations in the convolutional layers or the post-processing steps on the bounding boxes are the most time consuming. The resulting Pareto frontiers can be found in Fig. 5. The large jump in FLOPs in both frontiers happens when the head is no longer applied to the first feature map. The difference is that in the FLOPs frontier the remaining layers still have 8 anchors left, while in the bounding boxes frontier there are only 5 anchors left.

Table 4 compares the result of retraining this FLOPs frontier configuration to the baseline and a RetinaNet version with only 1 anchor in each layer as reported in the original paper. Not only does our pruned configuration achieve 4.1% better accuracy, it can also reduce the computational cost of the head by 75% compared to the RetinaNet (s3a3) baseline and by 68% compared to RetinaNet (s1a1). This means that our method is able to reduce the FLOPs of the entire RetinaNet model by 44% while only losing 1.8% accuracy which is 3× better

**Table 4**

Accuracy, FLOPs and inference time for RetinaNet variants on COCO test-dev2017. Our pruned model can reduce the FLOPs by 75% compared to the baseline (s3a3) while being much more accurate than the smallest original RetinaNet head (s1a1). Inference times are measured on an Nvidia GTX 1080.

| Model              | AP <sub>.5-.95</sub> | FLOPs<br>head | FLOPs<br>total | inf time |
|--------------------|----------------------|---------------|----------------|----------|
| RetinaNet(s3a3)    | 36.9                 | 129B          | 224B           | 137 ms   |
| RetinaNet(s1a1)    | 31.0                 | 98B           | 193B           | 99 ms    |
| Pruned             | 33.9                 | 31B           | 126B           | 79 ms    |
| Pruned(+retrained) | 35.1                 | <b>31B</b>    | 126B           | 79 ms    |

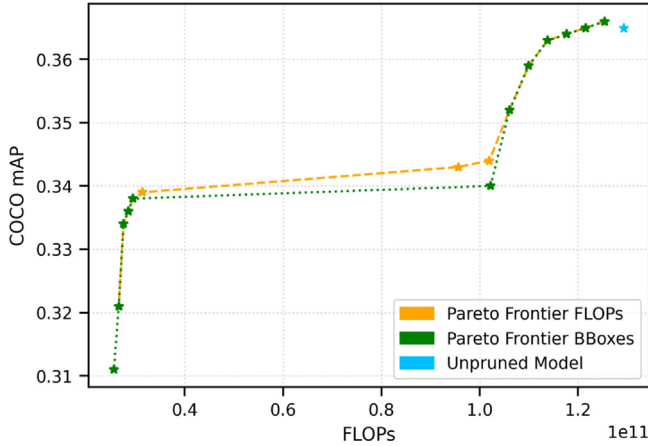


Fig. 5. Accuracy and number of FLOPs for pruned anchor configurations of RetinaNet on COCO val2017. The Pareto frontiers are optimized for the number of FLOPs or the number of bounding boxes.

compression and 4.1% better accuracy than the default way of scaling anchors, *i.e.* RetinaNet(s1a1), which only reduces the FLOPs by 14% while also decreasing the accuracy by 5.9%.

These results also illustrate the importance of adapting the anchor pruning search to the object detection architecture; without allowing an entire layer to be dropped it is impossible to significantly decrease the number of FLOPs.

The training configurations used for RetinaNet are based on the MMDetection (Chen et al., 2019) RetinaNet baseline configuration with ResNet50 backbone and 1 $\times$  learning rate schedule. All models are trained using SGD for 12 epochs with an initial learning rate of 0.01 which is decreased to  $10^{-3}$  on epoch 8 and  $10^{-4}$  on epoch 11, the weight decay is  $10^{-4}$  and the momentum 0.9. The input images are resized to 800 pixels on the shortest side, and random horizontal flipping is used as data augmentation.

### 3.7. MobileNetV2-SSDLite

To further demonstrate the generalization of our approach we also conducted experiments on an SSD object detector with a more compact backbone. It is generally assumed that compact models such as MobileNet (Sandler et al., 2018) and ShuffleNet (Hu et al., 2018) are harder to prune due to their already compact layers (Zhu and Gupta, 2017). For anchor pruning we show that the used backbone is not of importance as the pruning happens in the object detection head. The only important factor in anchor pruning is the placement of the original anchors. Fig. 6 shows the Pareto Frontiers for MobileNetV2-SSDLite and the original SSD model. Both frontiers follow a similar trend line, indicating that anchor pruning is backbone agnostic. The main difference between the two trend lines is the slightly worse accuracy degradation on the left side in the MobileNet frontier. This is however explained by the fact that MobileNetV2-SSDLite model has only 3 anchors in the first layer compared to 4 in the original SSD model.

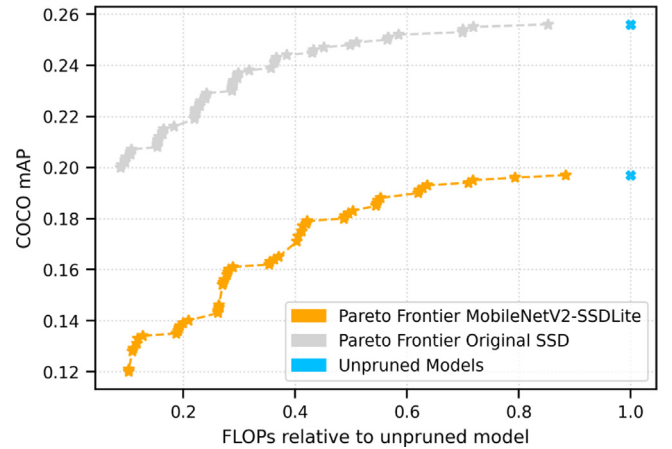


Fig. 6. Accuracy and number of FLOPs relative to the unpruned models for MobileNetV2-SSDLite and the original SSD on COCO val2017. Both frontiers have a similar trend line, indicating that anchor pruning is backbone agnostic.

### 3.8. PASCAL VOC

For completeness and to further demonstrate the generalization of our approach, we also evaluate our approach on the PASCAL VOC (Everingham et al., 2010) dataset. As is common, we use the combined datasets of VOC2007 and VOC2012. To reproduce the SSD baseline in the same way as the original paper, we change the anchor scales to the values specific for PASCAL VOC as defined in Liu et al. (2016). The training configuration remains identical to the one used for COCO, with the exception that it is run for twice as many epochs.

The Pareto frontier for SSD on this dataset can be found in Fig. 7. The reported mAP is with an IoU threshold of 0.50 as is custom for evaluating on this dataset. It can be seen that anchor pruning with retraining can achieve similar mAP as the baseline with more than 50% FLOPs reduction.

As SSD uses different anchor shape initialization for PASCAL VOC compared to COCO, we also included an ‘overanchored’ model that has the same initial anchors as in the experiments of Section 3.5. For both PASCAL VOC and COCO pruning from the same ‘overanchored’ model results in an improvement over the baseline models. This demonstrates that anchor pruning can not only eliminate the number of anchors as parameters but also the shapes of the anchors when used with an ‘overanchored’ detection head.

Compared to the results on COCO, pruning SSD on PASCAL VOC shows an additional interesting property; removing certain anchors immediately improves accuracy, even before fine-tuning or retraining. This can be explained by looking at the pruned anchors. For example, the most accurate configuration prunes, among others, all anchors in the last layer. In SSD the last layer of the head operates on a  $1 \times 1$  feature map while the previous layer operates on a  $3 \times 3$  feature map. When anchors from both these layers make a prediction about the same large object, the anchor from the layer with the larger feature map has more spatial information which can lead to more accurate bounding box predictions. The example image in Fig. 8 illustrates this: while the anchor from the last layer is more confident in its prediction, the anchor from the previous layer is more accurate in predicting the bounding box shape. Depending on the network structure in the backbone and neck, removing all anchors on the last feature map may make this layer obsolete and reduce the computational complexity of the entire network even further.

## 4. Conclusion

In this paper, we proposed a novel pruning method for one-stage anchor-based object detection models: anchor pruning. We show that

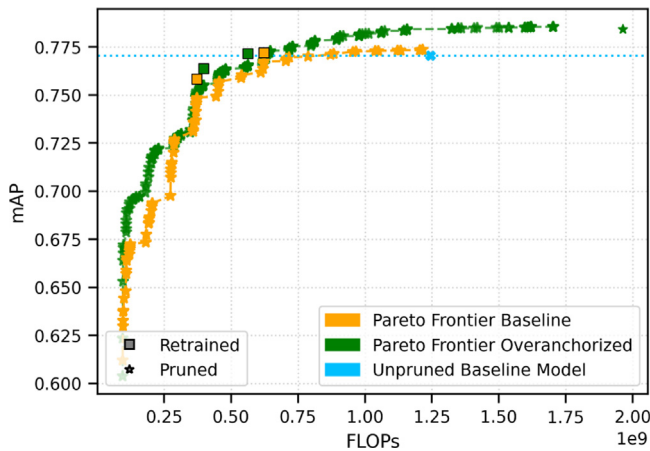


Fig. 7. Accuracy and number of FLOPs for anchor pruning configurations in the Pareto frontiers for the SSD baseline and ‘overanchored’ model on PASCAL VOC. Note how pruning certain anchors results in improved accuracy compared to the baseline, even before any fine-tuning or retraining.



Fig. 8. An example illustrating how predictions made by a large anchor can be more confident but less accurate (a) than predictions made by a smaller anchor in an earlier layer (b).

in most object detectors many anchors are redundant and that pruning those anchors followed by a fine-tuning or retraining step can increase accuracy while simultaneously reducing the computational cost. Through a simple yet effective search algorithm, we provide a Pareto frontier of anchor configurations, allowing model designers to trade accuracy for performance when resources are constrained. We demonstrate the effects of pruning anchors extensively on the SSD and RetinaNet object detectors and the MS COCO and PASCAL VOC datasets. We also show that through pruning an ‘overanchored’ model we avoid tuning hyperparameters related to the initial shapes of the anchors. Given its effectiveness to make object detection models more efficient, we hope that anchor pruning will become part of the design process for modern one-stage object detectors.

#### CRedit authorship contribution statement

**Maxim Bonnaerens:** Conceptualization, Methodology, Software, Visualization, Writing – original draft. **Matthias Freiberger:** Writing – review & editing. **Joni Dambre:** Supervision, Conceptualization, Methodology, Writing – review & editing.

#### Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

#### Acknowledgments

This research received funding through the imec.icon project cRE-Ative and the Research Foundation Flanders (FWO-Vlaanderen) under Grant G006718N and 1S47820N.

#### References

- Cai, Z., Vasconcelos, N., 2018. Cascade r-cnn: Delving into high quality object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6154–6162.
- Cai, L., Zhao, B., Wang, Z., Lin, J., Foo, C.S., Aly, M.S., Chandrasekhar, V., 2019. MaxpoolNMS: getting rid of NMS bottlenecks in two-stage object detectors. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9356–9364.
- Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C.C., Lin, D., 2019. MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint [arXiv:1906.07155](https://arxiv.org/abs/1906.07155).
- Deng, L., Li, G., Han, S., Shi, L., Xie, Y., 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108 (4), 485–532.
- Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A., 2010. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vis.* 88 (2), 303–338.
- Girshick, R., Fast r-cnn. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448.
- He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778.
- Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint [arXiv:1503.02531](https://arxiv.org/abs/1503.02531).
- Hu, J., Shen, L., Sun, G., 2018. Squeeze-and-excitation networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7132–7141.
- Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al., 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7310–7311.
- Ke, W., Zhang, T., Huang, Z., Ye, Q., Liu, J., Huang, D., 2020. Multiple Anchor Learning for Visual Object Detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10206–10215.
- Law, H., Deng, J., 2018. Cornernet: Detecting objects as paired keypoints. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 734–750.
- Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2016. Pruning filters for efficient convnets. arXiv preprint [arXiv:1608.08710](https://arxiv.org/abs/1608.08710).
- Li, S., Yang, L., Huang, J., Hua, X.-S., Zhang, L., 2019. Dynamic anchor feature selection for single-shot object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 6609–6618.
- Liao, M., Shi, B., Bai, X., 2018. Textboxes++: A single-shot oriented scene text detector. *IEEE Trans. Image Process.* 27 (8), 3676–3690.
- Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017a. Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2117–2125.
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P., 2017b. Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2980–2988.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: Common objects in context. In: European Conference on Computer Vision. Springer, pp. 740–755.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. Ssd: Single shot multibox detector. In: European Conference on Computer Vision. Springer, pp. 21–37.
- Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T., 2018. Rethinking the value of network pruning. arXiv preprint [arXiv:1810.05270](https://arxiv.org/abs/1810.05270).
- Luo, W., Li, Y., Urtasun, R., Zemel, R., 2016. Understanding the effective receptive field in deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 4898–4906.
- Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788.
- Redmon, J., Farhadi, A., 2017. YOLO9000: better, faster, stronger. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7263–7271.
- Redmon, J., Farhadi, A., 2018. Yolov3: An incremental improvement. arXiv preprint [arXiv:1804.02767](https://arxiv.org/abs/1804.02767).
- Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems. pp. 91–99.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520.



- Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y., 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint [arXiv:1312.6229](https://arxiv.org/abs/1312.6229).
- Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint [arXiv:1409.1556](https://arxiv.org/abs/1409.1556).
- Tan, M., Pang, R., Le, Q.V., 2020. Efficientdet: Scalable and efficient object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10781–10790.
- Tian, Z., Shen, C., Chen, H., He, T., 2019. Fcos: Fully convolutional one-stage object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 9627–9636.
- Verucchi, M., Brilli, G., Sapienza, D., Verasani, M., Arena, M., Gatti, F., Capotondi, A., Cavicchioli, R., Bertogna, M., Solieri, M., 2020. A systematic assessment of embedded neural networks for object detection. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vol. 1. IEEE, pp. 937–944.
- Wang, J., Chen, K., Yang, S., Loy, C.C., Lin, D., 2019. Region proposal by guided anchoring. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2965–2974.
- Yang, T., Zhang, X., Li, Z., Zhang, W., Sun, J., 2018. Metaanchor: Learning to detect objects with customized anchors. In: Advances in Neural Information Processing Systems. pp. 320–330.
- Zhang, X., Wan, F., Liu, C., Ji, R., Ye, Q., 2019. Freeanchor: Learning to match anchors for visual object detection. In: Advances in Neural Information Processing Systems. pp. 147–155.
- Zhang, S., Wen, L., Bian, X., Lei, Z., Li, S.Z., 2018. Single-shot refinement neural network for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4203–4212.
- Zhang, S., Zhu, X., Lei, Z., Shi, H., Wang, X., Li, S.Z., 2017. S3fd: Single shot scale-invariant face detector. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 192–201.
- Zhong, Y., Wang, J., Peng, J., Zhang, L., 2020. Anchor box optimization for object detection. In: The IEEE Winter Conference on Applications of Computer Vision. pp. 1286–1294.
- Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y., 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv preprint [arXiv:1702.03044](https://arxiv.org/abs/1702.03044).
- Zhu, M., Gupta, S., 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint [arXiv:1710.01878](https://arxiv.org/abs/1710.01878).