

# Micro-YOLO: Exploring Efficient Methods to Compress CNN based Object Detection Model

Lining Hu<sup>a</sup> and Yongfu Li<sup>b</sup>

Department of Micro-Nano Electronics, MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University, China

**Keywords:** Object Detection, YOLO, MobileNets, Depthwise Separable Convolution, Model Compression, Prune.

**Abstract:** Deep learning models have made significant breakthroughs in the performance of object detection. However, in the traditional models, such as Faster R-CNN and YOLO, the size of these networks make it too difficult to be deployed on embedded mobile devices due to limited computation resources and tight power budgets. Hence, we propose a new light-weight CNN based object detection model, Micro-YOLO based on YOLOv3-Tiny, which achieves a significant reduction in the number of parameters and computation cost while maintaining the detection performance. We propose to replace convolutional layers in the YOLOv3-tiny network with the Depth-wise Separable convolution (*DSCov*) and the mobile inverted bottleneck convolution with squeeze and excitation block (*MBCov*), and design a progressive channel-level pruning algorithm to minimize the number of parameters and maximize the detection performance. Hence, the proposed Micro-YOLO network reduces the number of parameters by  $3.46\times$  and multiply-accumulate operation (MAC) by  $2.55\times$  while slightly decreases the *mAP* evaluated on the COCO dataset by 0.7%, compared to the original YOLOv3-tiny network.

## 1 INTRODUCTION


The accelerated growth in the deep learning field has greatly promoted the development of the object detection with its widespread applications in face detection, autonomous driving, robot vision and video surveillance (Borji et al., 2019; Pan et al., 2020). With the vigorous development in object detection, there are several deep convolutional neural network models proposed in the recent years, .e.g. R-CNN, SSD, and YOLO (Girshick et al., 2014; Liu et al., 2016; Redmon and Farhadi, 2018). However, as the network becomes more complicated, the size of these models continues to increase, which makes it increasingly difficult to deploy these models on embedded devices in real life (Cheng et al., 2017). Therefore, it is of vital importance to develop an efficient and fast object detection model to reduce the parameter size without affecting the object detection quality.


The goal of object detection is to detect objects of a certain class (such as humans, animals, or cars) in digital images (Borji et al., 2019). One of the most famous object detection network is “You Only Look

Once” (YOLO) architecture. After years of improvement for YOLO, it has evolved into the fourth generation, YOLOv4 architecture (Bochkovskiy et al., 2020). It achieves average precision (*AP*) of 43.5% (65.7% *AP*<sub>50</sub>) for the MS COCO dataset at a real time speed of 65 frames per second (*FPS*) on Tesla V100(Bochkovskiy et al., 2020). However, it contains more than 60 million parameters and requires to perform more than 107 billion floating number multiplications when processing an image. Besides, the faster version of YOLOv3, the previous version of YOLOv4, YOLOv3-tiny is proposed where its parameters and multiplication requirements have reduced by  $7.5\times$  and  $13\times$ , respectively (Redmon and Farhadi, 2018). The new model has achieved 33.1% *mAP* with 220FPS on Titan X. However, it remains challenging to deploy this model for several embedded devices.

In this work, we propose a lightweight version of the objection detection model, Micro-YOLO, which is based on YOLOv3-tiny architecture (Redmon and Farhadi, 2018). We proposed three effective methods to optimize the Micro-YOLO architecture. The key contributions of our work are as follows:

- 1) We propose to replace the standard convolutional layers (*Conv*) in the YOLOv3-tiny network with depth-wise separable convolutions (*DSCov*) and

<sup>a</sup>  <https://orcid.org/0000-0003-3506-7873>

<sup>b</sup>  <https://orcid.org/0000-0002-6322-8614>

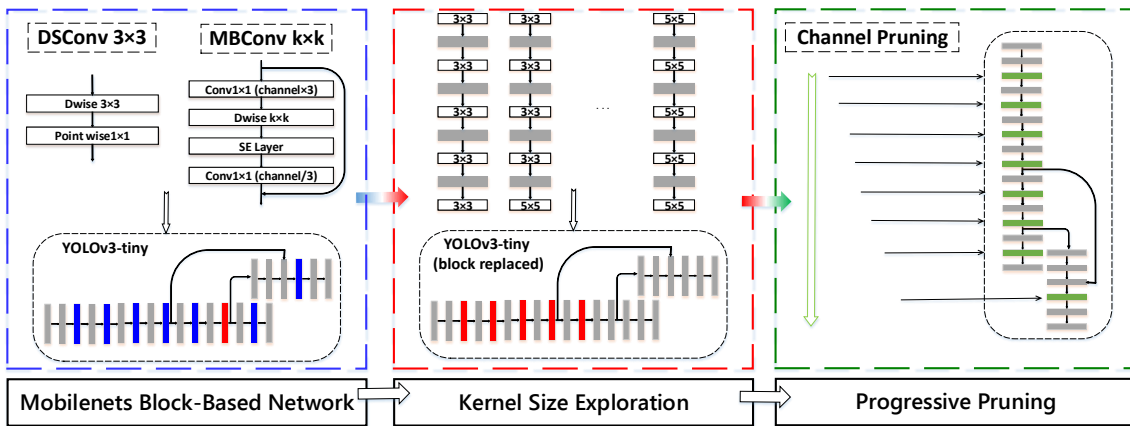


Figure 1: The proposed optimization techniques adopted in Micro-YOLO.

inverted bottleneck convolution with squeeze and excitation block (*MBCConv*), reducing the weight parameters with slightly degrading the detection accuracy.

- 2) We explore and identify the optimal kernel sizes in *MBCConv* to achieve the best trade-off between the weight parameters and detection accuracy on the Micro-YOLO architecture.
- 3) We propose a progressive pruning algorithm to perform a coarse-grained pruning on the *DSConv* and *MBCConv* layers, which further reduce the weight parameters with slightly degrading the detection accuracy. After pruning, we further decrease the size to 1.92M parameters and the computation cost to 0.87GMAC with 3.1% *mAP* degradation.

The rest of this paper is organized as follows. Section 2 provides a understanding of the state-of-arts model compression techniques and its evaluation methods and problem statement. Section 3 provides details of our proposed Micro-YOLO network and its model compression methods. Section 4 discusses the experimental setup and result and followed by the comparison with state-of-the-art works. We conclude our work in Section 5.

## 2 PRELIMINARIES

### 2.1 Model Compression Techniques for Object Detection Networks

As the family of object detection networks continues to become more complicated, it is important to reduce the weight parameters and computational cost. The model compression methods are categorized into low-rank factorization, knowledge distillation, pruning,

and quantization (Fernandez-Marques et al., 2020), where pruning has shown to be an effective method in reducing the network complexity by removing redundant parameters (Cheng et al., 2017).

To address the object detection network problem, there are several state-of-art works techniques to reduce the number parameters in YOLO architecture. (Huang et al., 2018) developed the YOLO-lite network, where batch normalization layer is removed from YOLOv2-tiny to speed up the object detection. This network has achieved a *mAP* of 33.81% and 12.26% on PASCAL VOC 2007 and COCO dataset, respectively. (Wong et al., 2019) created a highly compact network, YOLO-nano, which is a 8-bit quantized model based on YOLO network and is optimized on PASCAL VOC 2007 dataset. This network has achieved 3.18M model size and 69.1% *mAP* on the PASCAL VOC 2007 dataset.

### 2.2 Evaluation Methods

We evaluate the effectiveness of object detection network based on three aspects: model size, computation cost and accuracy performance on the COCO dataset (Lin et al., 2014).

**Definition 1 (Model Size).** Model size is defined as the number of parameters in a neural network, which is the sum of trainable elements in each layer. It is formulated as follows:

$$\text{Model Size} = \sum_{i=1}^N l_i, \quad (1)$$

where  $l_i$  denotes the number of trainable elements in the  $i$ -th layer and  $N$  represents the total number of layers in the neural network.

**Definition 2 (Computation Cost).** We define Computation Cost as the number of multiply-accumulate operations (MACs) which is the count of operation units

in which the product of two numbers is computed and that product is added to an accumulator.

**Definition 3** (Mean average precision (mAP)). The most common evaluation method for object detection is “Average Precision” (AP), which is defined as the average detection precision under different recalls. Precision measures how accurate is the model predictions. Recall measures how good the model finds all the positives. mAP (mean average precision) is the average of AP. For COCO dataset, we evaluate the mAP on 80 categories.

### 2.3 Problem Formulation

With the above definitions, the problem is formulated as follows:

[Model Compression for Object Detection Problem]

Given an object detection neural network model, the objective is to utilize efficient compression schemes on the model to achieve small *Model Size* and *Computation Cost* while maintaining the network’s *mAP*.

## 3 OUR PROPOSED METHOD

As shown in Fig. 1, we propose three methods on the YOLOv3-tiny network and obtain a lightweight version of the network, named Micro-YOLO: (1) To reduce the convolutional network blocks in the YOLO network, we propose to replace the standard convolution (*Conv*) layers with two types of convolutional blocks: (a) the depth-wise separable convolution (*DSConv*) used in MobileNetV1 (Howard et al., 2017) and (b) mobile inverted bottleneck convolution with squeeze and excitation block (*MBCConv*) used in MobileNetV3(Howard et al., 2019); (2) We explore and identify the optimal kernel sizes in *MBCConv* to achieve the best trade-off between the weight parameters and detection accuracy on the network; (3) We propose a progressive structured pruning method to further shrink the *Model Size*.

### 3.1 MobileNets Block-based Network

To reduce the size of the network, we have explored alternative lightweight convolutional layers to replace the convolutional layers *Conv* in the YOLO network. The MobileNet networks (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019) adopt two lightweight convolutional layers (a) the Depth-wise separable convolution (*DSConv*) layer and (b) mobile inverted bottleneck convolution with squeeze and excitation block (*MBCConv*) layer. As shown in the

Fig. 2(a), *DSConv* layer performs two types of convolutions: (i) the depthwise convolution and (ii) the pointwise convolution, which can significantly reduce the *Model Size* and *Computation Cost* of the network (Howard et al., 2017). As shown in the Fig. 2(b), the structure of *MBCConv* is a  $1 \times 1$  channel expansion convolution followed by depthwise convolutions and a  $1 \times 1$  channel reduction layer. It utilizes squeeze and excitation block, which is a branch consisting of a global average pooling operation in the squeeze phase and two small *FC* layers in the excitation phase (Hu et al., 2019) between depthwise convolution and channel reduction layer. Since the number of output channels is not equal to the number of input channels, we remove the residual connection in *MBCConv*. *MBCConv* layer provides a compact representation at the input and output while expanding the input to a higher-dimensional feature space internally to increase the expressiveness of nonlinear transformations. Hence, the *MBCConv* layer provides a better compressed network without degrading the detection accuracy as compared to the *DSConv* layer.

To evaluate the *Model Size* amongst these layers, the number of parameters in the *Conv* ( $N_s$ ), in the *DSConv* ( $N_{ds}$ ), and in the *MBCConv* ( $N_{mb}$ ) can be computed with (2), (3) and (4), respectively.

$$N_s = k^2 \times C_{in} \times C_{out}, \quad (2)$$

$$N_{ds} = k^2 \times C_{in} + 1 \times 1 \times C_{in} \times C_{out}, \quad (3)$$

$$N_{mb} = C_{in} \times \alpha C_{in} \times 1 \times 1 + k^2 \times \alpha C_{in} + 2 \times \alpha C_{in} \times \alpha C_{in} / \beta + \alpha C_{in} \times C_{out}, \quad (4)$$

where  $k$  denotes kernel size,  $C_{in}$  denotes number of input channels,  $C_{out}$  denotes number of output channels,  $\alpha$  and  $\beta$  denotes expansion factor and reduction factor in *MBCConv*, respectively.

The *Computation Cost* amongst these layers, i.e. the *Conv* layer ( $C_s$ ), the *DSConv* layer ( $C_{ds}$ ), and the *MBCConv* layer ( $C_{mb}$ ) can be expressed with the following (5), (6), (7), respectively.

$$C_s = \frac{1}{2} k^2 \times W \times H \times C_{in} \times C_{out}, \quad (5)$$

$$C_{ds} = \frac{1}{2} (k^2 \times W \times H \times C_{in} + W \times H \times C_{in} \times C_{out}), \quad (6)$$

$$C_{mb} = \frac{1}{2} (W \times H \times C_{in} \times \alpha C_{in} + k^2 \times W \times H \times \alpha C_{in} + 2 \times \alpha C_{in} \times \alpha C_{in} / \beta + W \times H \times \alpha C_{in} \times C_{out}), \quad (7)$$

where  $k$  denotes kernel size,  $C_{in}$  denotes number of input channels,  $C_{out}$  denotes number of output channels,  $W$  and  $H$  denote width and height of feature maps,  $\alpha$  and  $\beta$  denotes expansion factor and reduction factor in *MBCConv*, respectively.

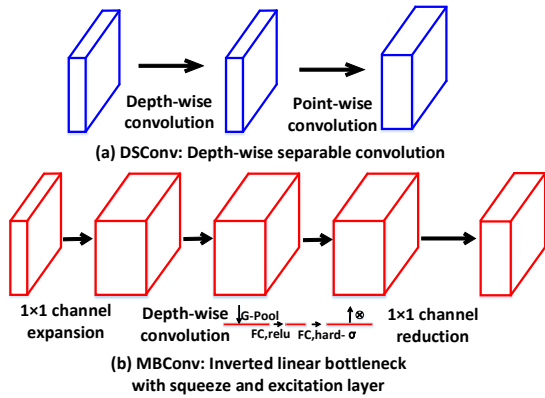


Figure 2: Two types of convolutions used in our work.

### 3.2 Kernel Size Exploration

To further reduce the weight parameters in the convolutional layer without compromising the accuracy, we propose a kernel size optimization technique. Most of the traditional convolutional neural network design uses  $3 \times 3$  convolutional kernel (Howard et al., 2017; Sandler et al., 2018). Similarly, YOLOv3-tiny network also uses convolution kernel size of  $3 \times 3$  in the *Conv* layers. However, the emergent of network architecture search algorithms changed the situation. For example, (Cai et al., 2019) highlighted that the first few convolutional layers prefer to using smaller kernel sizes while the deep convolutional layer prefers to using larger kernel sizes. Furthermore, recent works on network exploration (Tan and Le, 2019) have shown a similar result that the combination of multiple kernel sizes leads to better detection accuracy. Hence, it is necessary to explore the optimization space between the use of different convolutional kernel size and the *mAP* of our proposed Micro-YOLO network. The details of our experiment will be discussed in Section 4.

### 3.3 Progressive Channel Pruning

After finalizing the architecture of our proposed Micro-YOLO network, we can further reduce the weight parameters by using the pruning technique. In our proposed work, we have adopted coarse-grained pruning because the *DSConv* and *MBConv* layers are mostly composed of  $1 \times 1$  kernel size, which left minimal room for fine-grained pruning. (Liu et al., 2019) indicates that the pruned architecture itself, rather than a set of inherited “important” weights, is more crucial to the efficiency in the final model, which suggests that in some cases pruning can be useful as an architecture search paradigm. Hence, we proposed a progressive pruning method to search for a “thinner”

Algorithm 1: Progressive Channel Pruning Algorithm.

**Input:** The original network structure  $Net(C_1 \dots C_N)$ .

**Output:** The pruned network structure  $\hat{Net}$ .

```

1: for  $i = 1$  to  $N$  do
2:   Train  $Net$  for 20 epochs;
3:   Evaluate  $mAP_{origin}$  of  $Net$ ;
4:    $OldC_i = C_i$ ,  $mAP_{old} = mAP_{origin}$ ;
5:   repeat
6:      $NewC_i = OldC_i - 1/16C_i$ ;
7:      $OldC_i = NewC_i$ ;
8:     Initialize a new network  $\hat{Net}(C_1 \dots C_N)$ ;
9:     Train  $\hat{Net}$  for 20 epochs;
10:    Evaluate  $mAP_{new}$  of  $\hat{Net}$ ;
11:     $mAP_{old} = mAP_{new}$ 
12:  until  $mAP_{new} < mAP_{old} - 0.5\%$ 
13: end for

```

architecture in the modified network. The details of the proposed progressive channel pruning algorithm are shown in Algorithm 1. We first train the original network  $Net$  and evaluate the  $mAP_{original}$  before pruning (Lines 2-3). The numbers of current pruned convolutional layer channels  $OldC_i$  are recorded (Line 4). During the pruning of convolutional layer  $i$ , we reduce the number of output channels by 1/16 each time since this pruning step balances the pruning speed and accuracy. Note that when the output channel of layer  $i$  is pruned, the corresponding input channel of layer  $i+1$  also needs to be pruned. Then the number of channels of convolutional layer  $i$  is updated, and a new network ( $\hat{Net}$ ) with the reduced number of channels is initialized (Lines 6-8).  $\hat{Net}$  is retrained for 20 epochs to evaluate the new  $mAP_{new}$  (Lines 9-11). The pruning procedure for layer  $i$  is repeated until  $mAP_{new}$  is 0.5% lower than the original  $mAP_{origin}$  since our experiment shows a threshold of 0.5% ensure that channel pruning will not decrease the detection accuracy severely (Line 12). Then, we begin to prune the next convolutional layer until all the convolutional layers are pruned, and the pruned network is returned.

## 4 EXPERIMENTAL RESULTS

We implemented and evaluated our Micro-YOLO network using Python programming language with Pytorch (Paszke et al., 2017) library on a 2.50GHz 12 cores Xeon Intel Linux machine, 128GB memory, and 2 Nvidia GTX 2080Ti graphics cards.

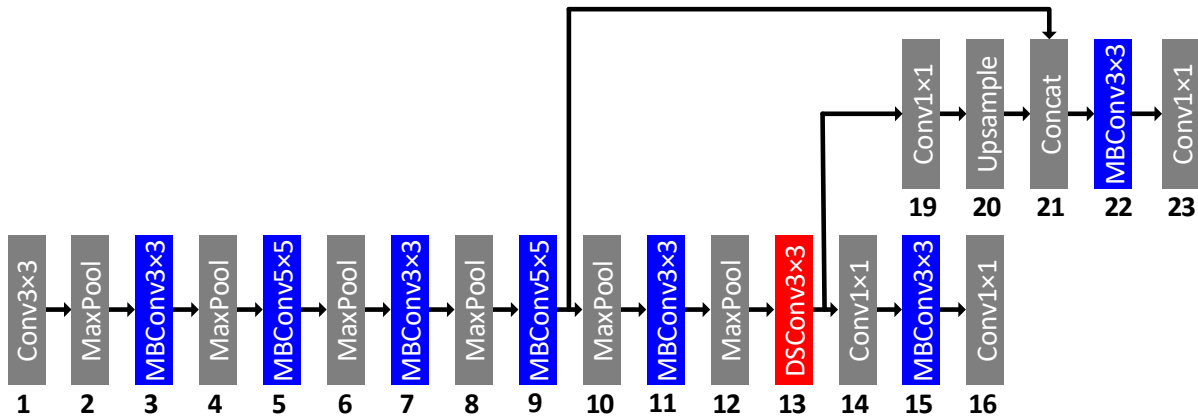


Figure 3: Our proposed object detection neural network architecture.  $MBConv_{k \times k}$  denotes mobile inverted bottleneck convolution with squeeze and excitation block with kernel size  $k \times k$ ,  $DSCConv_{k \times k}$  denotes depth-wise separable convolution with kernel size  $k \times k$ .

#### 4.1 Micro-YOLO Network Optimization

In our proposed Micro-YOLO network, the choice of convolution type in each layer and kernel size in the convolutional layer have a great influence on the detection accuracy. Thus, we conduct experiments to determine the architecture for the Micro-YOLO network.

##### 4.1.1 The Choice of Convolution Types

As discussed in Section 3.1, there are great differences among the number of parameters in the  $Conv$ ,  $DSCConv$  and  $MBConv$  layers. As shown in Table 1, we compute the number of parameters required for different layer types and different input channels with the same kernel size according to (2)-(7). Note that the number of output channels is twice the number of input channels. As shown in the last two columns of the table, the number of parameters used in  $MBConv$  and  $DSCConv$  layers are significantly smaller than  $Conv$  layer.

To understand the impact of different convolution types on *Model Size*, *Computation Cost* and *mAP*, we replace  $Conv$  of YOLOv3-tiny with our proposed strategies. Table 2 shows *Model Size*, *Computation Cost* of networks composed of different convolution types and *mAP* evaluated on COCO dataset. As shown in the table, networks that with only  $DSCConv$  layers have far smaller *Model Size* and *Computation Cost* compared to networks consists of  $MBConv$  layers only. However, using the  $MBConv$  layer is more effective in maintaining the *mAP* while  $DSCConv$  can be applied to reduce the number of parameters. Hence, it is necessary to choose an optimal trade-off

between *Model Size* and *mAP* of the network.

As shown in Tables 1 and 2, the increase in the number of input channel and convolutional layers leads to the increase of *Model Size*. For example, in the YOLOv3-tiny model, the 10th, 12th, and 14th layers have a total weight parameters of 6.63M, which accounts for 74.95% of the entire network. We use  $DSCConv$  in the 12th layer and  $MBConv$  in the remaining layers since the 12th layer contains the largest amount of parameters. This leads to the *Model size* reduction by  $3.46 \times$  while the *mAP* only degrades by 1.7%. Hence, the final form of our proposed Micro-YOLO network is shown in Fig. 3.

##### 4.1.2 Kernel Size Exploration

As discussed in Section 3.2, the choice of kernel size is very essential to improve *mAP*. Therefore, we choose the 3rd, 5th, 7th, 9th, and 11th layers, which are layers before the detection part of YOLOv3-tiny, to explore the effect of different kernel sizes on those layers. For each layer, we choose kernel size from  $3 \times 3$  and  $5 \times 5$ , thus leading to  $2^5=32$  different permutations and combinations. To save our training time, we train each experiment for 20 epochs from scratch and find the best combination of these permutations and combinations. As shown in Figure 4, among the 32 kinds of combinations, the quality of the networks which interleaving  $3 \times 3$  and  $5 \times 5$  kernel sizes is the best. Thus, this indicates that the best *mAP* is achieved by using convolution kernels of size 3,5,3,5,3 in the 3rd, 5th, 7th, 9th, 11th layers, respectively.

Table 1: Number of parameters required for different convolution types and different input channels with the same kernel size  $3 \times 3$ .

No. of Input Channels	No. of Parameters			Redution Multiples <sup>1</sup>	
	<i>Conv</i>	<i>DSCConv</i>	<i>MBCConv</i>	<i>DSCConv</i>	<i>MBCConv</i>
16	4,068	656	3,888	$7.02 \times$	$1.19 \times$
32	18,432	2,336	14,688	$7.89 \times$	$1.25 \times$
64	73,728	8,768	57,024	$8.41 \times$	$1.29 \times$
128	294,912	33,920	224,640	$8.69 \times$	$1.31 \times$
256	1,179,648	133,376	891,648	$8.84 \times$	$1.32 \times$
512	4,718,592	528,896	3,552,768	<b><math>8.92 \times</math></b>	<b><math>1.33 \times</math></b>

<sup>1</sup> Reduction Multiples denote the reduced multiples of parameters compared to standard convolution.

Table 2: The amount of parameters of the YOLOv3-tiny network composed of different convolution types.

Network	Model Size (M)	Computation Cost (G)	mAP %
YOLOv3-tiny	8.85	2.81	<b>33.1</b>
All <i>DSCConv</i> $3 \times 3$	<b>1.44</b>	<b>0.52</b>	24.6
All <i>DSCConv</i> $5 \times 5$	1.47	0.55	25.4
All <i>MBCConv</i> $3 \times 3$	6.45	2.08	30.4
All <i>MBCConv</i> $5 \times 5$	6.53	2.16	31.5

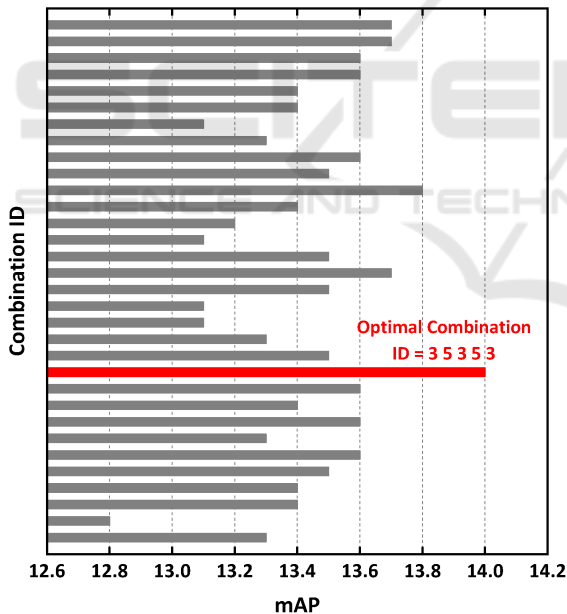


Figure 4: Kernel size exploration result. Different bars indicate different combinations of kernel sizes. For simplicity, we only show the optimal kernel size combination in red.

## 4.2 Pruning Results

To further compress our model, we have applied our progressive pruning algorithm for the first 7 convolution layers. The pruning results are presented in Table 3, where the  $x/16$  indicates the pruning step of each

layer. For example, the 3rd layer contains 32 channels, which we first prune 2 channels based on the  $1/16$  of the initial number of channels calculation. At the second step, we prune 4 channels, which is  $2/16$  of 32 channels. When pruning  $3/16$  of the initial number of channels, compared with the initial value, the *mAP* decreased by 1.4%, which is greater than 0.5%, then we stop pruning this layer and move on to the next layer.

As shown in Table 3, most of the convolution layers cannot be further pruned when we perform pruning on  $2/16$  of the number of channels. If we continue to perform pruning, the *mAP* starts to degrade significantly. Hence, the results shown in Table 3 has also confirmed our conjecture: As the depth of the network and the number of convolutional layer channels increase, the convolutional layer’s “tolerance” to pruning gradually increases, enabling us to prune more channels in deeper layers, such as 11th and 13rd layers. In particular, we even prune  $6/16$  of the number of channels, that is, 384 channels, in the 13th layer without decreasing *mAP*. However, in the 15th layer, we observe an exception situation where even  $1/16$  of the number of channels cannot be pruned. We suspect that the reason may be that this layer is too close to the detection layer.

## 4.3 Benchmark and Comparisons

We have made a comparison between our proposed Micro-YOLO against YOLO-nano(Huang et al., 2018), YOLO-lite(Wong et al., 2019) and YOLOv3-tiny (Redmon and Farhadi, 2018). We trained all of the networks from scratch for 500,200 batches, similar to the training method used in YOLOv3-tiny (Redmon and Farhadi, 2018). Table 4 illustrates the *Model size*, *Computation cost*, *mAP* on COCO datasets and *FPS* of YOLOv3-tiny, YOLO-lite, YOLO-nano and Micro-YOLO.

As compared with the YOLOv3-tiny network, the initial version of our Micro-YOLO has already

Table 3: Pruning results with progressive channel pruning algorithm.

Layer ID	Kernel Size	No. of Channels	$mAP$ %							
			Original	1/16	2/16	3/16	4/16	5/16	6/16	7/16
3	3	32	17	16.8	<b>16.8</b>	15.6	-	-	-	-
5	5	64	16.8	16.8	<b>16.4</b>	15.3	-	-	-	-
7	3	128	16.4	16.1	16.2	16.1	16	<b>16.2</b>	15	-
9	5	256	16.2	16.2	16	<b>15.9</b>	14	-	-	-
11	3	512	15.9	15.6	<b>15.5</b>	15	-	-	-	-
13	3	1024	15.5	15.5	15.4	15.5	15.6	15.6	<b>15.7</b>	14.7
15	3	512	<b>15.7</b>	15.0	-	-	-	-	-	-

Table 4: Model’s amount of parameters, computation cost,  $mAP$ , and latency of YOLOv3-tiny, YOLO-lite, YOLO-nano, and Micro-YOLO (original and pruned). The input size is  $416 \times 416$  for all networks.

Model	Model Size (M)	Computation Cost (GMAC)	$mAP$ % (CO-CO)	$mAP$ % (VOC 2007)	$mAP$ % (VOC FPS)
YOLOv3-tiny	8.85	2.81	33.1	-	313
YOLO-lite	0.46	0.93	12.3	33.6	378
YOLO-nano	3.18	3.49	14.5	69.1	240
Micro-YOLO	<b>2.56</b>	<b>1.10</b>	<b>32.4</b>	-	<b>328</b>
Micro-YOLO (pruned)	<b>1.92</b>	<b>0.87</b>	<b>29.3</b>	-	<b>357</b>

achieved a significant reduction of the parameters by  $3.46 \times$  and the number of operations by  $2.55 \times$  with slightly decrease of  $0.7\%$   $mAP$  on COCO dataset. After applying coarse-grained pruning technique, the Micro-YOLO has reduced the weight parameters by  $4.61 \times$  and computation cost by  $3.23 \times$  with a slight drop of  $3.8\%$   $mAP$  compared with YOLOv3-tiny. YOLO-lite model has a size of  $0.46M$  parameters and requires a computation cost of  $0.93GMAC$  and achieves  $12.3\%$   $mAP$  on the COCO dataset. YOLO-nano model has a size of  $3.18M$  parameters and requires a computation cost of  $3.49GMAC$  and achieve  $14.5\%$   $mAP$  and  $69.1\%$   $mAP$  on COCO and PASCAL VOC 2007 datasets, respectively, it’s because YOLO-nano is optimized based on the PASCAL VOC 2007 dataset, it does not perform very well on the COCO dataset. As for the latency, we re-evaluate the FPS of all the networks on a single Nvidia GTX 2080Ti graphics card. Our Micro-YOLO and pruned Micro-YOLO achieve 328 and 357 FPS respectively, second only to YOLO-lite. Since YOLO-nano is optimized based on the PASCAL VOC 2007 dataset, it does not perform very well on the COCO dataset.

## 5 CONCLUSIONS

In this paper, we explore several model compression methods and propose an improved object detection architecture, Micro-YOLO, based on YOLOv3-tiny. We analyze several types of convolutional layers, such as depth-wise separable convolution (*DSCov*) and inverted bottleneck convolution with squeeze and excitation block (*MBCov*), to determine the optimal layer for our Micro-YOLO network. We also explore the effect of different kernel sizes in these convolutional layers on Micro-YOLO performance. Furthermore, we propose a new progressive channel pruning method to minimize the number of parameters and computation costs with slightly  $mAP$  reduction of the original network. The Micro-YOLO only requires  $2.56M$  parameters and  $1.10GMAC$  of *Computation Cost* to achieve the  $mAP$  of  $32.4\%$  and  $328$  FPS, which is slightly lower than the original YOLOv3-tiny network. After applying the pruning technique, we can further reduce the number of parameters and computation cost to  $1.92M$  and  $0.87GMAC$  with  $mAP$  of  $29.3\%$  and  $357$  FPS. We also compare our work with other variety of YOLO-based networks for object detection and achieve promising results. We believe that our methodology to compress YOLOv3-tiny can be highly applicable to the future version of YOLO or other object detection models.

## ACKNOWLEDGEMENTS

This research is supported in part by the National Key Research and Development Program of China under Grant No. 2019YFB2204500 and in part by the Science, Technology and Innovation Action Plan of Shanghai Municipality, China under Grant No. 1914220370.

## REFERENCES

- Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv preprint arXiv:2004.10934*.
- Borji, A., Cheng, M.-M., Hou, Q., Jiang, H., and Li, J. (2019). Salient object detection: A survey. *Computational Visual Media*, 5(1):117–150.
- Cai, H., Zhu, L., and Han, S. (2019). Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.
- Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.
- Fernandez-Marques, J., Whatmough, P. N., Mundy, A., and Mattina, M. (2020). Searching for Winograd-aware Quantized Networks. *MLSys*.
- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE CVPR*, pages 580–587.
- Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for Mobilenetv3. *IEEE ICCV*, pages 1314–1324.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
- Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. (2019). Squeeze-and-Excitation Networks. *IEEE PAMI*, 5(1):117–150.
- Huang, R., Pedoeem, J., and Chen, C. (2018). YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. In *IEEE Big Data*, pages 2503–2510.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *ECCV*, pages 21–37.
- Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2019). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.
- Pan, M., Zhu, X., Li, Y., Qian, J., and Liu, P. (2020). MR-Net: A Keypoint Guided Multi-scale Reasoning Network for Vehicle Re-identification. In Yang, H., Papas, K., Leung, A. C., Kwok, J. T., Chan, J. H., and King, I., editors, *Neural Information Processing, ICONIP 2020*, volume 1332, pages 469–478.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.
- Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE CVPR*, pages 4510–4520.
- Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*.
- Wong, A., Famuori, M., Shafiee, M. J., Li, F., Chwyl, B., and Chung, J. (2019). Yolo nano: a highly compact you only look once convolutional neural network for object detection. *arXiv preprint arXiv:1910.01271*.