IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Mini-YOLOv3: Real-Time Object Detector for Embedded Applications

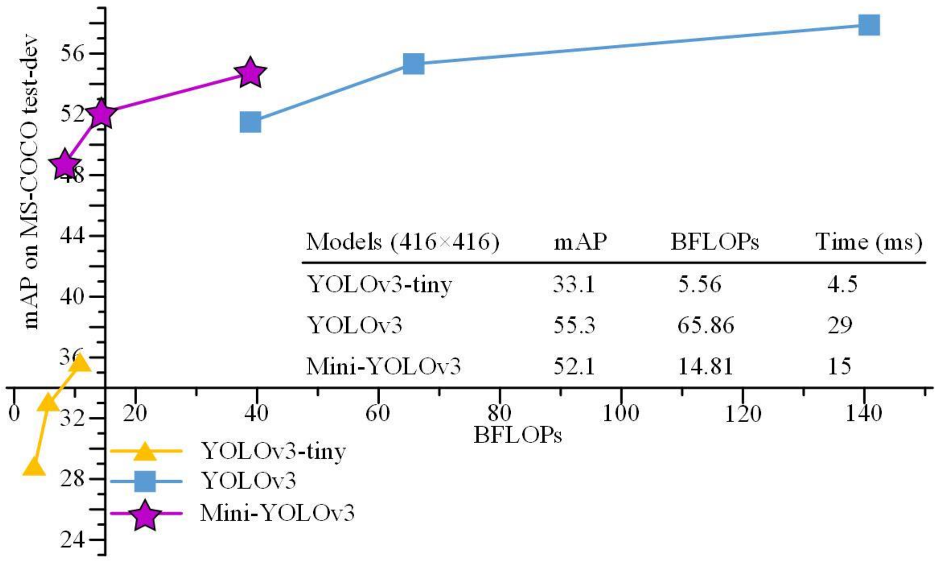
Qi-Chao Mao[1], Hong-Mei Sun[1,2], Yan-Bo Liu[1], Rui-Sheng Jia[1,2]
[1] College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China
[2] Shandong Province Key Laboratory of Wisdom Mine Information Technology, Shandong University of Science and Technology, Qingdao 266590, China

Corresponding author: Hong-Mei Sun (e-mail: shm0221@163.com); Rui-Sheng Jia (e-mail: jrs716@ 163.com).

This work was supported in part by the Natural Science Foundation of Shandong Province, China (ZR2018MEE008), in part by the Key Research and Development Program of Shandong Province, China (2017GSF20115).

ABSTRACT Real-time scene parsing through object detection running on an embedded device is very challenging, due to limited memory and computing power of embedded devices. To deal with these challenges, we redesign a lightweight network without notably reducing detection accuracy. Based on the Darknet-53, we use depth separable convolutions and pointwise group convolutions to reduce the parameter size of the network. A feature extraction backbone network with a parameter size of only 16 percent of darknet-53 is constructed. Meanwhile, in order to compensate for the degradation of accuracy, we have added a Multi-Scale Feature Pyramid Network based on a simple U-shaped structure to improve the performance of multi-scale object detection, which called it Mini-YOLOv3. It has smaller model size and fewer trainable parameters and floating point operations (FLOPs) in comparison of YOLOv3. We evaluate Mini-YOLOv3 on MS-COCO benchmark dataset; The parameter size of Mini-YOLOv3 is only 23% of YOLOv3 and achieves comparable detection accuracy as YOLOv3 but only requires 1/2 detect time, Specifically, Mini-YOLOv3 achieves mAP-50 of 52.1 at speed of 67 fps.

INDEX TERMS Real-Time object detector; Embedded Applications; Convolutional neural network (CNN); YOLOv3.



Models (416×416)	mAP	BFLOPs	Time (ms)
YOLOv3-tiny	33.1	5.56	4.5
YOLOv3	55.3	65.86	29
Mini-YOLOv3	52.1	14.81	15

FIGURE 1. Billion floating point operations (BFLOPs) versus accuracy (mAP) on MS-COCO benchmark dataset. By lightweight processing, Mini-YOLOv3 achieves comparable detection accuracy as YOLOv3 but only requires a little more floating point operations as YOLOv3-tiny. Such performance is very competitive in embedded applications.

I. INTRODUCTION

With the rapid development of technologies such as smart phones, autopilot cars [1], and drones [2], more and more embedded devices have been endowed with computer vision function. For example, Autopilot technology require vehicles are able to sense environment, parse scene and react accordingly; Artificial Intelligence (AI) retouching and other functions in the smartphone need to accurately locate the object position in the image; Object detection might be the most common one that is adopted as a basic functional module for scene parsing in embedded applications, and hence it has been the area of increasing interest.

Object detection is a classic task in the field of computer vision. It is the basic premise for advanced visual tasks such as Video Content Recognition (VCR) and Image Understanding (IU) [3]. Some traditional object detection methods based on manual features have been gradually eliminated with the advent of neural networks because of their low accuracy and poor environmental adaptability [4~9]. Because of the growth of computing power and the dedicated deep learning chips and the availability of large-scale labelled samples (e.g., ImageNet [10,11], PASCAL VOC [12] and MS-COCO [13]), Convolutional Neural Network(CNN) has been extensively studied due to its fast, scalable and end to-end learning framework. Some object detection methods based on Region Proposal Networks (RPN) such as R-CNN [14], Fast R-CNN [15], Faster R-CNN [16] and Mask R-CNN [17] constantly update the highest accuracy of object detection. But in order to maintain high accuracy, these methods must rely on powerful GPU computing power. However, such improvement in accuracy with heavy computational cost may not be helpful to face the challenge in many real world applications that require real-time performance carried out in a computationally limited platform. In addition, with the latest advances in technologies such as autonomous driving, the ability of models to perform object detection in real time is also an essential factor. An efficient and fast object detector is key to the success of autonomous driving [18], Augmented Reality (AR) devices [19], and other smart device. Recently, some fast single-stage object detectors have been proposed, such as SSD [20], DSSD [21], YOLO series models [22][23][24], RetinaNet [25] ,etc. Among them, YOLOv3 might be the most popular object detector in practical applications as the detection accuracy and speed are well balanced. Despite that, YOLOv3 still requires heavy computational cost and large run-time memory footprint to maintain good detection performance; it brings high computation overhead and power consumption to embedded devices. Only reducing the size of the model can run the object detection method on the embedded device. The YOLO series, SSD series and other networks have lightweight versions called YOLO-tiny [22], YOLO-Lite [26] tiny SSD [27], but the detection accuracy of the networks is greatly reduced. Therefore, how to reduce the model size and floating point operations (FLOPs) without notably reducing detection accuracy becomes an urgent problem to be solved when deploying object detectors on embedded device. The most popular method is to reduce the parameters and model size of the network by redesigning a more efficient network. For example, SqueezeNet [28], MobileNet [29], ShffuleNet [30], etc., these methods can maintain detection accuracy to a great context with significantly reduces the model size and floating point operations (FLOPs).

Based on YOLOv3, we have redesigned a lightweight network structure without notably reducing detection accuracy. A feature extraction backbone network with a parameter size of only 16 percent of darknet-53 is constructed. Meanwhile, in order to compensate for the degradation of accuracy, we have added a Multi-Scale Feature Pyramid network based on a simple U-shaped structure to improve the performance of multi-scale object detection, which we called it Mini-YOLOv3. It has smaller model size and fewer trainable parameters and floating point operations (FLOPs) in comparison of original YOLOv3. We evaluate Mini-YOLOv3 on MS-COCO benchmark dataset; It is worth noting that Mini-YOLOv3 achieves ~77.6% decrease of FLOPs, ~76.7% decline of parameter size and comparable detection accuracy as YOLOv3. Specifically, Mini-YOLOv3 achieves mAP-50 of 52.1 at speed of 67 fps.

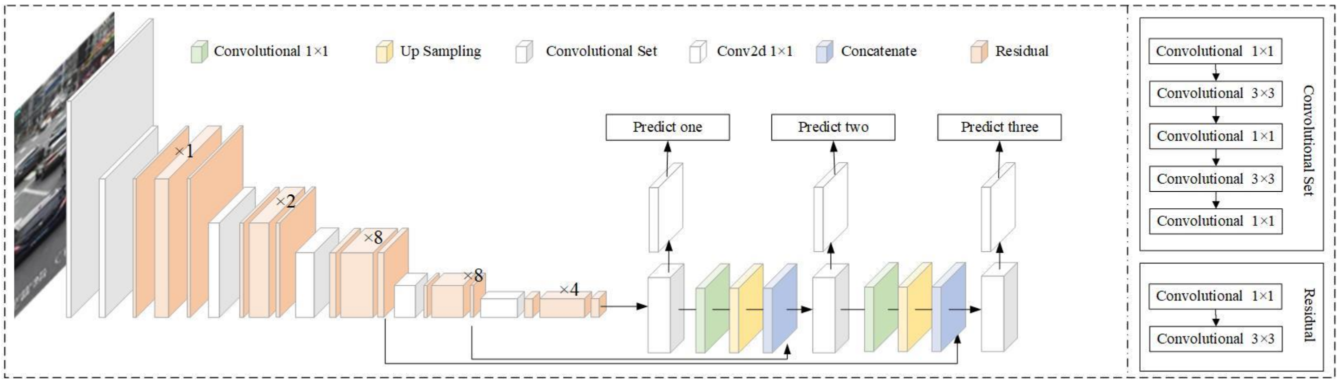


FIGURE 2. **Structure detail of YOLOv3.It uses Darknet-53 as the backbone network and uses three scale predictions.**

II. RELATED WORK

YOLOv3 [24] is an object detector proposed by Joseph et al., and it takes the detection procedure as a regression task. This method increases the speed of detection and accepts input pictures of different sizes. YOLOv3 use Darknet-53[24] for performing feature extraction. Darknet-53[24] is much more powerful than Darknet-19[22] but still more efficient than ResNet-101[31] or ResNet-152[31]. YOLOv3 uses multi-scale prediction, which means it is detected on multiple scale feature maps. For this reason, the accuracy of target detection is improved. Its structure detail is shown in Figure 2.

YOLOv3 testing process is as follows:

Step1. Input the image and scale the image to the standard size.

Step2. Divide the input image into 13*13, 26*26 52*52 grids of three scales. If the center point of an object falls in the grid unit, the grid unit predicts the object.

Step3. Using *k*-means clustering to determine our bounding box priors on each grid unit. There are three clusters on each grid unit. Because of the three scales, there are a total of 9 clusters per grid unit.

Step4. Input the image into the network for feature extraction. The model first produces a feature map of 13*13 on a small scale.

Step5. The 13*13 small-scale feature map is first subjected to convolutional set and 2 times upsampling, and then connected to the 26*26 feature map and output the prediction result.

Step6. the 26*26 feature map output by step5 is subjected to convolutional set and 2 times upsampling, then connected to the 52*52 feature map and output the prediction result.

Step7. Fusion of features from three scale predictive outputs, Afterward, using a probability score as a threshold to filter out most anchors with low scores. Then using Non Maximum Suppression (NMS) [32] for post-processing, leaving more accurate boxes.

YOLO v3 can achieve real-time detection on the high performance computer by means of the powerful computing capability of the GPU. Because the performance of embedded devices is far less than that of high Performance Computer, it is often impossible to achieve real-time applications.

III. MINI-YOLOv3

We have redesigned the lightweight backbone network to increase the speed of detection. Considering that the reduction of the backbone network layer has a negative impact on accuracy, as a complement, we propose a Multi-Scale Feature Pyramid Network (MSFPN) to enhance the feature extraction capabilities. Its detail is shown in Figure 3. Our method uses the backbone and the MSFPN to extract features from the input image, and then produces predicts bounding boxes based on the learned features, followed by the Non-Maximum Suppression (NMS) to produce the results. For the MSFPN, there are three modules, the Concat module responsible for connecting each feature of backbone network, the Encoder-Decoder module responsible for generating multi-scale features, and the Feature Fusion module responsible for aggregating features. Below we will detail the various modules of the network.

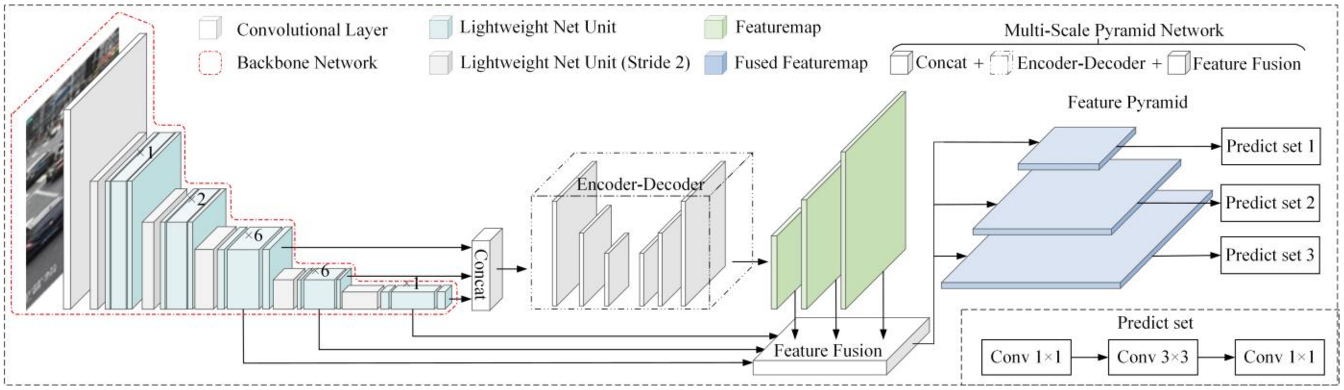


FIGURE 3. An overview of the Mini-YOLOv3. Mini-YOLOv3 uses the lightweight backbone and the Multi-Scale Feature Pyramid Network (MSFPN) to extract features from the input image. In MSFPN, Concat model fuses three feature maps of the backbone to generate the base feature. Encoder-Decoder generates a group of multi-scale features, Feature Fusion model aggregates three feature maps of the backbone and the group of multi-scale features into a feature pyramid.

A. BACKBONE

In order to improve the speed of the method, we decided to simplify the network. We try to improve the efficiency of the convolutional by using square connections. First, we define the unit of the network:

We changed the standard convolution to depthwise separable convolutions according to Google "Xception" [33]. It is a form of factorized convolutions that factorize a standard convolution into two layers; the first layer is a depthwise convolution [29]. The depthwise convolution applies a single filter to each input channel. The second layer is a 1×1 convolution called a pointwise convolution [34]. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size. We can see that by the following. First, we define the number of input channels M, the number of output channels N the kernel size $D_K \times D_K$ and the feature map size $D_F \times D_F$.

Standard convolutions have the computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{1}$$

Depthwise separable convolutions cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{2}$$

which is the sum of the depthwise and 1×1 pointwise convolution. By expressing convolution as a two-step process of filtering and combining, we get a reduction in computation of:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \tag{3}$$

It can be seen that the calculation by the depth separable convolution is $\frac{1}{N} + \frac{1}{D_K^2}$ of the ordinary convolution.

After that, we use the shortcut connection as shown in Figure 4 (c). Because Resnet and Densenet have adopted residual representations and shortcut connection, and fully proved the effectiveness [31] [35]. In Resnet, each residual uses a stack of 3 layers. The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions. Considering that we use depthwise convolution, if we first reduce the dimension and then increase (restore) the dimension, it will greatly weaken the expression ability of the model. Therefore, we adopt the opposite strategy. We define C_i as the input channel and define C_i' as the output channel. We use 1×1 convolution to increase to $t * C_i$ ($t = 4$ in the experiment), and then reduce the dimension to C_i', leaving the 3×3 layer an inverted bottleneck with larger input/output dimensions.

However, the calculation of pointwise convolution generated by the above operation is huge. So we use 1x1 pointwise group convolution (in experiment, group parameter $g = 4$) [30]. However, it also brings an obvious problem: As shown in Figure 5(a): if multiple group convolutions stack together, outputs from a certain channel are only derived from a small fraction of input channels. We added Channel Shuffle after 1 pointwise group convolutions. The purpose of shuffle operation is to enable cross-group information flow for multiple group convolution layers as shown in Figure 5(b). So our network unit is shown in Figure 4(d). Figure 4(e) is the down-sampling layer. We use the depthwise separable convolution of stride = 2 for down-sampling, without the short cut connecting the input and output features.

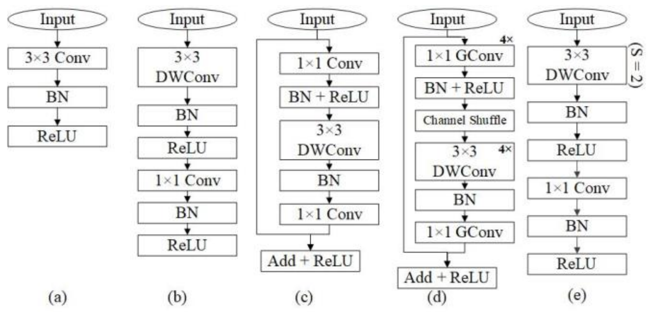


FIGURE 4. (a)Standard convolutional layer with batchnorm and ReLU; (b) Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU; (c) Add a shortcut connection to b; (d) Our network unit(stride=1) block; (e) Our network unit (stride=2).

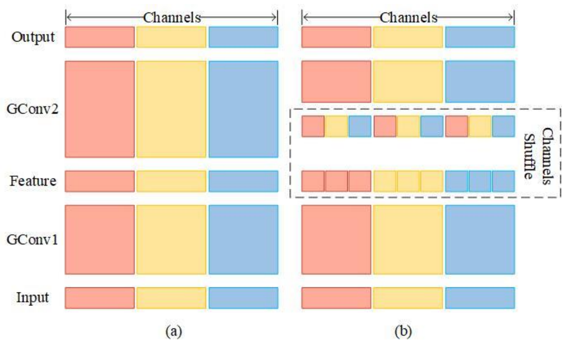


FIGURE 5. (a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; (b) G Conv using channel shuffle. Where G Conv stands for group convolution.

We redesigned our backbone network base on Darknet-53. The network structure is shown in table I. Our network consists of the network unit shown in Figure 4(d) (e), except that the first and last layers of the network are standard convolutions.

TABLE I THE DETAILS OF CONV LAYER

	Type	Filter	Channel	Stride	Featuremap
	Conv 2d	3×3	32	1	416×416
	Conv DW	3×3	16	2	208×208
1×	G Conv	1×1	16	1	
	Conv DW	3×3	16	1	
	G Conv	1×1	16	1	208×208
	Conv DW	3×3	32	2	104×104
2×	G Conv	1×1	128	1	
	Conv DW	3×3	128	1	
	G Conv	1×1	32	1	104×104
	Conv DW	3×3	64	2	52×52
6×	G Conv	1×1	256	1	
	Conv DW	3×3	256	1	
	G Conv	1×1	64	1	52×52
	Conv DW	3×3	128	2	26×26
6×	G Conv	1×1	512	1	
	Conv DW	3×3	512	1	
	G Conv	1×1	128	1	26×26
	Conv DW	3×3	256	2	13×13
1×	G Conv	1×1	1024	1	
	Conv DW	3×3	1024	1	
	G Conv	1×1	256	1	13×13
	Conv 2d	1×1	1024	1	13×13

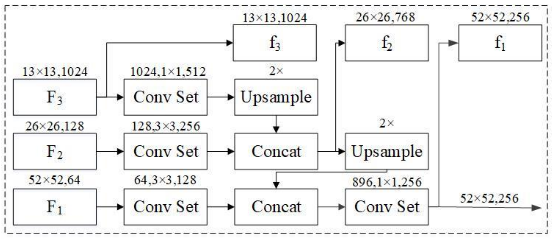


FIGURE 6. Structural details of Concat module. F$_1$, F$_2$, F$_3$ are feature maps of different scales of the backbone, as input to the Concat module; Example: 896, 1×1, 256: input channels: 896, Kernel size: 1×1, output channel: 256.

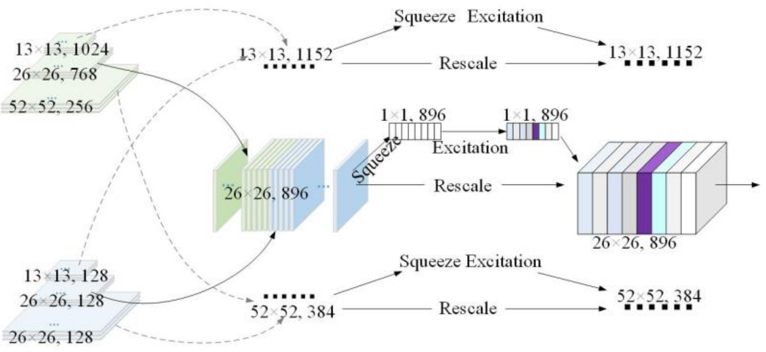


Figure 7. Structural details of Feature Fusion Module.

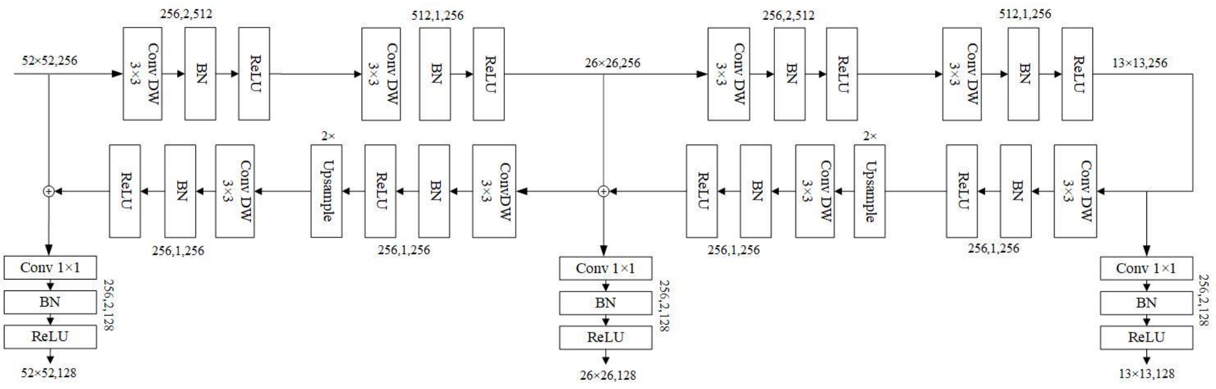


Figure 8. Illustration of Encoder-Decoder. 256, 1, 256: input channels: 256, Stride: 1, output channel: 256; ⊕: ele-wise sum.

B. MULTI-SCALE FEATURE PYRAMID NETWORK

The purpose of the Multi-Scale Feature Pyramid Network is to strengthen the features of the backbone network and generate a more efficient multi-scale feature pyramid. Concat model fuses three feature maps of the backbone to generate the base feature. Encoder-Decoder generates a group of multi-scale features, Feature Fusion model aggregates three feature maps of the backbone and the group of multi-scale features into a feature pyramid. Below we describe the three modules in detail.

1) CONCAT MODULE

Concat model fuse features from three different levels in backbone, which are crucial to constructing the final feature pyramid. The input to the Concat module is the three feature maps from the backbone network, and we upsample them to rescale the depth features to the same scale before concatenating. Its structural details are shown in Figure 6.

2) ENCODER-DECODER MODULE

The role of the Encoder-Decoder is to generate feature maps with three scales. Its structure is shown in Figure 8. We built a thin U-shape to construct the Encoder-Decoder. In the encoder, in order to generate reference-set of feature maps, we use a series of 3×3 convolutional layers with stride 2 to downsample the input feature map. The decoder is a series of 3 × 3 convolutional layers with stride 1. Of course, there are upsample and elementwise sum operation. Finally, we add a 1×1 convolutional layer at the branch to enhance the feature representation and keep feature smoothness [35].

3) FEATURE FUSION MODULE

Feature Fusion module aims to aggregate the multiscale features generated by Encoder-Decoder and the features f$_1$, f$_2$, f$_3$ (Marked in Figure 6) from three different levels in backbone into a feature pyramid as shown in Figure 7; The first stage of Feature fusion module is to concatenate the multiscale features generated by Encoder-Decoder and f$_1$, f$_2$, f$_3$ with equivalent scales along channel dimension. It can be presented as $U=[U_1U_2\cdots U_i]$, Then We introduce a channel-wise attention module, using SE attention to aggregate features in an adaptive way [36]. The structural details are shown in Figure 7.

In SE, we first embed the global information, that is, we adopt the Global Average Pooling for feature blocks after fusion to squeeze global spatial information into a channel descriptor. As in the Squeeze in Figure 8. We define $z \in R^C$, where z is generated by shrinking U through spatial dimensions $H \times W$, the c-th element of z is calculated by:

$$z_c = F_{sq}(u_c) = \frac{1}{W \times H}\sum_{i=1}^{W}\sum_{j=1}^{H} u_c(i,j) \qquad (4)$$

To make use of the information aggregated in the squeeze operation, we follow it with a second operation, which aims to fully capture channel-wise dependencies, called excitation:

$$s = \mathrm{F}_{ex}(z,W) = \sigma(W_2\delta(W_1 z)) \qquad (5)$$

Where σ refers to ReLU function, δ refers to sigmoid function, $W_1 \in R^{\frac{C}{r}\times C}$, $W_2 \in R^{C\times\frac{C}{r}}$, r is reduction ratio, The final output of the block is obtained by rescaling the transformation output U with the activations:

$$X = F_{scale}(U_i^c, s_c) = s_c \cdot U_i^c \tag{6}$$

Where X is the final output of the SE block, $F_{scale}(U_i^c, s_c)$ refers to the multi-scale feature pyramid, each of the features is enhanced or weakened by SE module.

C. LOSS FUNCTION

As with YOLOv3, multi-part loss function is shown in Equation (7):

$$\begin{aligned} Loss = {} & \lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+ \\ & \lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}(2-w_i\times h_i)[(w_i-\hat{w}_i)^2+(h_i-\hat{h}_i)^2]- \\ & \sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[\hat{C}_i\log(C_i)+(1-\hat{C}_i)\log(1-C_i)]- \\ & \lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[\hat{C}_i\log(C_i)+(1-\hat{C}_i)\log(1-C_i)]- \\ & \sum_{i=0}^{S^2}1_i^{obj}\sum_{c\in classes}^{S^2}[\hat{p}_i(c)\log(p_i(c))+(1-\hat{p}_i(c))\log(1-p_i(c))] \end{aligned} \tag{7}$$

Where, the first line is the coordinate loss of the bounding box; The second line is the loss of the height and width of the bounding box; The third line is the loss of confidence in the bounding box of the existing object; The fourth line is the bounding box confidence loss of the object that does not exist. The fifth line is the classification loss of the cell in which the object exists. s^2 is the number of cells, B is the number of bounding boxes predicted by each grid, C is the number of classes, and p is the probability. Where 1_i^{obj} denotes if object appears in cell i and 1_{ij}^{obj} denotes that the j-th bounding box predictor in cell i is "responsible" for that prediction. λ_{coord} and λ_{noobj} are parameters that control the stability of the training.

IV. EXPERIMENT

In this section, our backbone network is pre-trained on the ImageNet 2012 dataset, and then the entire network is trained on the MS-COCO training set and performed on the MS-COCO test-dev set. In the experiment, we set the parameter g in all pointwise group convolutions to 4, in the SE block, the reduction ratio is set to 16, as for the input size, we can follow the original YOLOv3, that is, 320×320, 416 x416, 608×608. Using k-means clustering to determine 9 bounding box priors on each grid unit. Finally, we use the soft-NMS with a linear kernel for post processing at the end of the network, leaving more accurate boxes.

A. EXPERIMENTS ON BENCHMARK

In Table II, we compare Mini-YOLOv3 to state-of-art detectors on MS-COCO test-dev. The information involved includes the test results, the backbone network and the detection time. As can be seen from table 2, 1-stage object detector is significantly faster than the two-stage detector. It is worth noting that the speed of Mini-YOLOv3 is 1/2 of that of YOLOv3 416 [24], although the accuracy is slightly reduced. We consider that the main reason why the accuracy can be maintained is MSRPN. Some existing feature pyramids are simply constructed from the layers of the backbone designed for object classification task. However, in MSRPN, we first use Concat model to fuse three feature maps of the backbone, and then feed it into Encoder-Decoder module and Feature Fusion module to extract multi scale features. The feature map generated in MSRPN is more representative. Obviously, the speed of SSD300 [20], RefineDet320 [25], YOLOv2 [23] can be comparable to Mini-YOLOv3 but the accuracy is lower. Although Mask R-CNN [17], Cascade R-CNN [37], Corner Net [38], etc. have higher precision than Mini-YOLOv3, Mini-YOLOv3 has a speed of 20.3×, 9.4×, 15.2× compared to them. Some qualitative examples of our object detection results on MS-COCO data set are shown in Figure 10.

TABLE II DETECTION ACCURACY COMPARISONS IN TERMS OF MAP PERCENTAGE ON MS COCO TEST-DEV SET

	Method	Backbone	Avg. Precision, IOU			Avg. Precision, Area			Time
			0.5:0.95	0.5	0.75	small	medium	large	
Two-stage	Fast R-CNN[15]	VGG-16	20.5	39.9	19.4	4.1	20.0	35.8	2000ms
	Faster R-CNN[16]	VGG-16	21.9	42.7	-	-	-	-	142ms
	Faster R-CNN Resnet101[31]	ResNet-101	27.2	48.4	-	-	-	-	200ms
	R-FCN[39]	ResNet-101	29.9	51.9	-	10.8	32.8	45.0	111ms
	Faster R-CNN w FPN[40]	Res101-FPN	36.2	59.1	39.0	18.2	39.0	48.2	167ms
	Mask R-CNN[17]	ResNeXt-101	39.8	62.3	43.4	22.1	43.2	51.2	303ms
	Fitness-NMS[41]	ResNet-101	41.8	60.9	44.9	21.5	45.0	57.5	200ms
	Cascade R-CNN[40]	Res101-FPN	42.8	62.1	46.3	23.7	45.5	55.2	143ms
One-stage	SSD300[20]	VGG-16	25.1	43.1	25.8	6.6	25.9	41.4	23ms
	SSD512[20]	VGG-16	28.8	48.5	30.3	10.9	31.8	43.5	45ms
	DSSD321[22]	ResNet-101	28.0	46.1	29.2	7.4	28.1	47.6	105ms
	RetinaNet400[42]	ResNet-101	31.9	49.5	34.1	11.6	35.8	48.5	81ms
	RefineDet320[25]	VGG-16	29.4	49.2	31.3	10.0	32.0	44.4	25ms
	Corner Net[38]	Hourglass	40.5	57.8	45.3	20.8	44.8	56.7	227ms
	YOLOv2[23]	DarkNet-19	21.6	44.0	19.2	5.0	22.4	33.5	15ms
	YOLOv3 608[24]	DarkNet-53	33.0	57.9	34.4	18.3	35.4	41.9	50ms
	YOLOv3 416[24]	DarkNet-53	31.0	55.3	31.7	14.1	34.1	46.4	29ms
	Mini-YOLOv3	Proposed Net	29.8	52.1	30.8	13.2	33.1	45.2	15ms

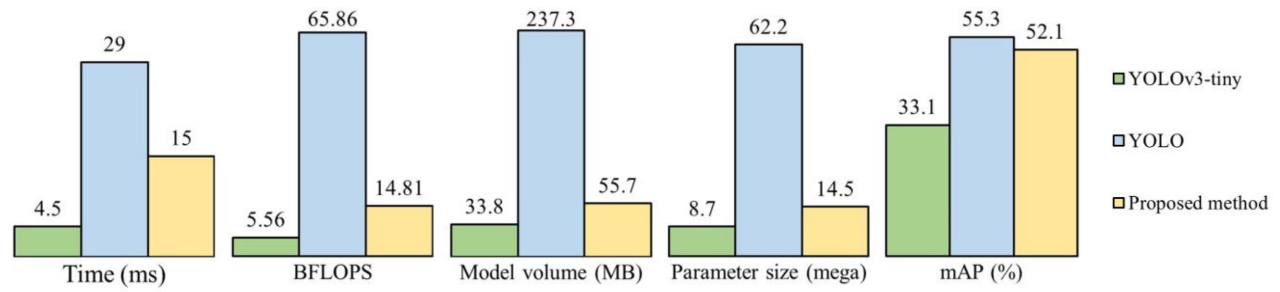


Figure 9. Comparison of YOLOv3, YOLOv3-tiny and Mini-YOLOv3 in Time, BLOPs, Model volume, Parameter size and mAP score when input size is 416×416.

TABLE III ABLATION STUDY OF MINI-YOLOV3 ON MS-COCO VAL SET

Module								
Pointwise group convolution:(g=4)	Channel Shuffle	Concat	Encoder-Decoder	Feature Fusion	AP,0.5:0.95	AP small	AP medium	AP larage
					24.8	12.3	29.3	36.4
√					24.0	11.9	28.4	35.0
√	√				26.1	12.7	30.7	38.8
√	√	√			27.8	13.0	34.2	40.7
√	√	√	√		29.9	13.8	34.8	46.1
√	√	√	√	√	30.4	14.7	35.0	46.9

B. MODEL ANALYSIS

Mini-YOLOv3 is a lightweight version of YOLOv3. Similarly, with the birth of YOLOv3, there is also a lightweight version called YOLOv3-tiny [25], It is a tiny version of YOLOv3, and is much faster but less accurate. We compare the detection performance of the above three models on MS-COCO dataset in Figure 1 and Figure 9. As shown in Figure 1 and Figure 9, YOLOv3 achieves the best detection results but requires the most FLOPs in the meantime. In contrast, Mini-YOLOv3 with little more parameters than YOLOv3-tiny is able to obtain suboptimal detection results that are comparable with YOLOv3. Obviously, Mini-YOLOv3 is much better than YOLOv3-tiny in detection accuracy. Such results imply that Mini-YOLOv3 is very competitive in embedded applications.

C. ABLATION STUDY

Since our network is made up of multiple modules, and the backbone network is also composed of multiple operations, so we need to verify its effectiveness for the final performance. Since the MS-COCO test dev contains too much data, our ablation studies were carried out in the MS-COCO val set.

1) POINTWISE GROUP CONVOLUTION:

As can be seen from the first and third rows of table III, thanks to pointwise group convolution with channel Shuffle, all results have been improved. In the experiment, we tried to change the value of g to observe its effect on the result. If the group number equals 1, no pointwise group convolution is involved. From Table IV, we see that models with group convolutions ($g > 1$) consistently perform better than the counterparts without pointwise group convolutions ($g = 1$). And when $g = 4$, get the peak AP = 30.4

TABLE IV AP VS. NUMBER OF GROUPS G

	AP,0.5:0.95			
Mini-YOLOv3	$g = 1$ 28.9	$g = 2$ 29.3	$g = 4$ 30.4	$g = 8$ 29.9

2) CHANNEL SHUFFLE:

The purpose of shuffle operation is to enable cross-group information flow for multiple group convolution layers. The second and third rows of Table III compare the performance of Mini-YOLOv3 with/without channel shuffle. It is obvious that channel shuffle consistently boosts the result.

3) CONCAT MODULE:

In order to verify that the Concat module has a positive impact on the result of the model, we compare the performance of Mini-YOLOv3 with/without Concat module. On the one hand, we do not use the Concat module, directly input the last layer network feature map into the Encoder-Decoder module; On the other hand, we use Concat to fuse the feature maps of the three different depths of the backbone network and then pass them to the Encoder-Decoder. Through the third and fourth rows of table III, we observe a noticeable AP gain from 25.9 to 27.1.

4) ENCODER-DECODER:

The role of the Encoder-Decoder module is to generate multi-scale features. In order to verify the effectiveness of the Encoder-Decoder module, we compare the performance of Mini-YOLOv3 with/without Encoder-Decoder. First, we send the three feature maps generated by the backbone network directly to the Feature Fusion module. Second, as for the Encoder-Decoder module, we still use the multi-scale feature map generated by it. Then the performance has improved compared with the last operation, shown in the fourth and fifth rows in Table III.

5) FEATURE FUSION MODULE:

As shown in the fifth and sixth rows in Table III, When the Feature Fusion module was added, all boxes including small, medium and large become more accurate.

V. CONCLUSION

In this work, we propose a lightweight object detection method called Mini-YOLOv3. First, for the backbone, based on the Darknet-53, we use depth separable convolutions and pointwise group convolutions to reduce the parameter size of the network. We continue to adopt the residual structure, and adopt the method of first reducing the dimension and then increasing (restoring) the dimension. To suppress the boundary effect of pointwise group convolution, we add channel shuffle. We constructed a backbone network with a parameter size of only 16 percent of darknet53. Then we propose a Multi-Scale Feature Pyramid Network (MSFPN). In MSFPN, Concat model fuses three feature maps of the backbone to generate the base feature. Encoder-Decoder generates a group of multi-scale features, Feature Fusion model aggregates three feature maps of the backbone and the group of multi-scale features into a feature pyramid, called the Fusion Feature Pyramid feature, and we added a channel-wise attention module to aggregate features. Finally, we test Mini-YOLOv3 on the MS-COCO dataset. Mini-YOLOv3 achieves comparable detection accuracy as YOLOv3 but only requires 1/2 detect time, Specifically, on MS-COCO benchmark, Mini-YOLOv3 achieves mAP-50 of 52.1 and AP: 0.5: 0.95 of 29.8 at speed of 67 fps. Such performance is very competitive in embedded applications.

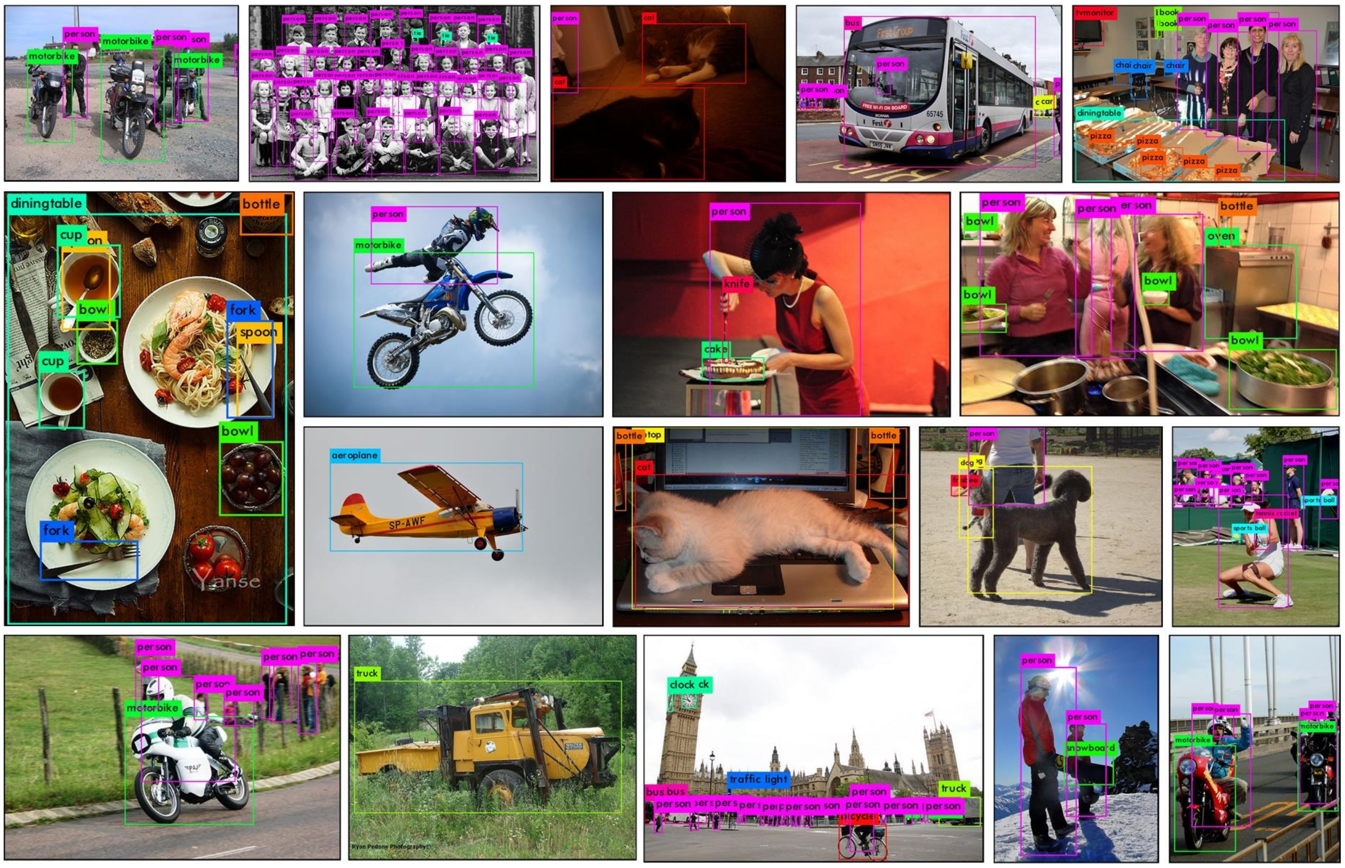


FIGURE 10. Some qualitative examples of our object detection results on MS-COCO data set.

REFERENCES

[1] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, 2016, [online] Available: https://arxiv.org/abs/1608.04644.

[2] I. Sa, S. Hrabar, P. Corke, "Outdoor flight testing of a pole inspection UAV incorporating high-speed vision", *Springer Tracts in Advanced Robotics*, vol. 105, pp. 107-121, 2014.

[3] Z.-Q. Zhao, P. Zheng, S.-t. Xu, X. Wu, "Object detection with deep learning: A review", *arXiv:1807.05511v1*, 2017.

[4] Paul Viola, Michael J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", *IEEE CVPR*, 2001.

[5] A. Leonardis, H. Bischof, "Robust Recognition Using Eigenimages", *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 99-118, 2000.

[6] N. Dalal, B. Triggs, "Histograms of Oriented Gradients for Human Detection", *Proc. IEEE Conf. Computer Vision and Pattern Recognition* , 2005.

[7] D. Lowe, "Object Recognition from Local Scale Invariant Features", *Proc. IEEE Int'l Conf. Computer Vision*, 1999.

[8] Y. Freund, R. Schapire, "Experiments with a New Boosting Algorithm", *Proc. 13th Int'l Conf. Machine Learning*, pp. 148-156, 1996.

[9] P. Felzenszwalb, D. McAllester, D. Ramanan, "A Discriminatively Trained Multiscale Deformable Part Model", *Proc. IEEE Conf. Computer Vision and Pattern Recognition* , 2008.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 248-255, 2009

[11] A. Krizhevsky, I. Sutskever, G. Hinton, "Imagenet classification with deep convolutional neural networks", *Proc. Adv. Neural Inf. Process. Syst.*, pp. 1106-1114, 2012

[12] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. Winn, A. Zisserma, "The PASCAL visual object classes challenge a retrospective", *Int. J. Comput. Vis.*, 2014.

[13] T.-Y. Lin et al., "Microsoft COCO: Common objects in context", *Proc. Eur. Conf. Comput. Vis.*, pp. 740-755, 2014.

[14] R. Girshick, J. Donahue, T. Darrell, J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, pp. 580-587, 2014.

[15] R. Girshick, "Fast R-CNN", *Proc. IEEE Int. Conf. Comput. Vis.*, 2015.

[16] S. Ren, K. He, R. Girshick, J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, pp. 91-99, 2015.

[17] K. He, G. Gkioxari, P. Dollár, R. Girshick, "Mask R-CNN", *Proc. IEEE Int. Conf. Comput. Vis*., pp. 2980-2988, 2017.

[18] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, J. Kindelsberger, L. Ding, S. Seaman et al., “Mit autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation,” *arXiv preprint arXiv:1711.06976*, 2017. 1

[19] O. Akgul, H. I. Penekli, Y. Genc, "Applying deep learning in augmented reality tracking", *Proc. 12th Int. Conf. Signal Image Technol. Internet Based Syst. (SITIS)*, pp. 47-54, 2016.

[20] W. Liu et al., "SSD: Single shot multibox detector", *Proc. Eur. Conf. Comput. Vis.*, pp. 21-37, 2016.

[21] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, A. C. Berg, Dssd: Deconvolutional single shot detector, 2017.

[22] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, "You only look once: Unified real-time object detection", *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, pp. 779-788, 2016.

[23] J. Redmon, A. Farhadi, YOLO9000: Better faster stronger, Dec. 2016, [online] Available: https://arxiv.org/abs/1612.08242.

[24] J. Redmon, A. Farhadi, YOLOv3: An incremental improvement, 2018, [online] Available: https://arxiv.org/abs/1804.02767.

[25] S. Zhang, L. Wen, X. Bian, Z. Lei, S. Z. Li, "Single-shot refinement neural network for object detection", *CVPR*, 2018.

[26] R. Huang, J. Pedoeem, C. Chen, "Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers", *IEEE International Conference on Big Data*, 2018.

[27] A. Wong, M. J. Shafiee, F. Li, B. Chwyl, Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection, 2018, [online] Available: https://arxiv.org/abs/1802.06488.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and; 0.5mb model size", *arXiv:1602.07360*, 2016.

[29] A. G. Howard et al., MobileNets: Efficient convolutional neural networks for mobile vision applications, Apr. 2017, [online] Available: https://arxiv.org/abs/1704.04861.

[30] M. G. Hluchyj, M. J. Karol, "ShuffleNet: An application of generalized perfect shuffles to multihop lightwave networks", *J. Lightwave Technol.*, vol. 9, pp. 1386-1397, 1991.

[31] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition", *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[32] A. Neubeck, L. Van Gool, "Efficient non-maximum suppression", *ICPR*, 2006.

[33] F. Chollet, "Xception: Deep learning with depthwise separable convolutions", *Proc. CVPR*, 2017.

[34] B.-S. Hua, M.-K. Tran, S.-K. Yeung, "Pointwise convolutional neural networks", *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 1-10, Jun. 2018.

[35] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids", *arXiv technical report*, 2014.

[36] J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks, Sep. 2017, [online] Available: https://arxiv.org/abs/1709.01507.

[37] Z. Cai, N. Vasconcelos, Cascade R-CNN: Delving into high quality object detection, 2017, [online] Available: https://arxiv.org/abs/1712.00726.

[38] H. Law, J. Deng, "CornerNet: Detecting objects as paired keypoints" in Computer Vision—ECCV, Cham, Switzerland:Springer, 2018.

[39] J. Dai, Y. Li, K. He, J. Sun, "R-FCN: Object detection via region-based fully convolutional networks", 2016

[40] T.-Y. Lin, P Dollar, R Girshick, K He, B Hariharan, S. Belongie, "Feature pyramid networks for object detection", *Computer Vision and Pattern Recognition*, vol. 1, no. 2, pp. 4, 2017.

[41] L. T. Smith, L. Petersson, "Improving object localization with fitness NMS and bounded iou loss", *CVPR* 2018.

[42] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár, "Focal loss for dense object detection", *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 2999-3007, 2017.