Image and Vision Computing 130 (2023) 104613

Contents lists available at ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

Single stage architecture for improved accuracy real-time object detection on mobile devices

Dan-Sebastian Bacea, Florin Oniga *

Computer Science Department, Technical University of Cluj-Napoca, Romania

ARTICLE INFO

Article history:
Received 22 September 2022
Received in revised form 8 December 2022
Accepted 11 December 2022
Available online 15 December 2022

Keywords:
Deep learning
Convolutional neural networks
Lightweight object detectors
YOLO
Mobile devices

ABSTRACT

YOLOv4-tiny is one of the most representative lightweight one-stage object detection algorithms. In this paper, we propose Mini-YOLOv4-tiny, an improved lightweight one-stage object detector based on the YOLOv4-tiny. Typical compression techniques address both memory optimization and computational complexity, but compromise model accuracy. For model compression and speedup, we selectively cut the width of the last convolutional layers. To increase model performance, we propose several improvements with minimal impact on memory and inference time. First, we replace the Leaky-RELU activation function with Mish and fine-tune the data augmentation parameters. To enlarge the receptive field of the network, we propose a modified Spatial Pyramid Pooling module with a reduced number of convolutional blocks and filters, thus adding fewer Floating Point Operations (FLOPs) during the inference process. We performed experiments on the PASCAL VOC and MS COCO datasets and evaluated the inference speed on NVidia Jetson Nano and Raspberry PI 4 (model B). The experimental results show that our methods achieve 72.1% mAP@0.5 and 23.4% mAP@ [0.5:0.95] on the PASCAL VOC and MS COCO datasets, achieving state-of-the-art results among lightweight object detectors. Compared with YOLOv4-tiny on MS COCO, our method has 37% fewer parameters and requires 19% fewer FLOPs, while improving the Intersection over Union (IoU) by 4.02% and the average precision (interval 0.50:0.95) by 2.8%. On PASCAL VOC, Mini-YOLOv4-tiny with an input size of 288 improves mAP@0.5 by 0.3%, while requiring 61% fewer FLOPS and running almost twice as fast.

1. Introduction

One of the most attractive and challenging topics in computer vision is object detection, which is widely adopted in tasks such as robot vision, industrial quality inspection, autonomous driving, object tracking, sports analysis, etc. Object detection accomplishes both localization and classification of objects in images. Traditional object detection methods [1,2] rely on features engineered by experts and classifiers, but they have high computational complexity and cannot meet real-time requirements. Since [3] achieved state-of-the-art (SOTA) results on [4], deep convolutional neural networks (DCNN) have been widely adopted as the go-to solution for object detection [5–15]. Initially, the focus was mainly on improving accuracy, leading to increasingly more complex models, while deployment on edge and mobile devices was not taken into consideration.

Two-stage detectors first predict regions in the image with a high chance of containing objects, then perform the classification and localization of the objects. Single-stage detectors directly predict the classification and localization in a single step. Two-stage models typically achieve higher performance than single stage models, but they are infeasible for deployment on computationally constrained devices. Lately, lightweight object detectors have attracted increasing interest, aiming to produce resource efficient object detection models. Single-stage object detectors such as the YOLO series [8,10–13] achieve a balance between speed and accuracy, the last of which (Scaled YOLOv4) reaches real-time performance and improved accuracy, when running on Graphics Processing Units (GPUs).

However, the execution speed of high performing YOLO models decreases significantly when deployed on edge devices, mainly due to the high computational volume. To address this issue, the lightweight YOLOv4-tiny [13] was proposed, which is one of the fastest object detection networks available today. However, due to its shallow architecture, the extracted features are relatively homogeneous, the accuracy is limited and the real-time performance on low computing power devices has room for improvement.

This paper introduces an efficient object detection neural network architecture designed for mobile devices that builds upon YOLOv4-tiny, called Mini-YOLOv4-tiny. The proposed model reduces the model

* Corresponding author.
E-mail addresses: Dan.Bacea@cs.utcluj.ro (D.-S. Bacea), Florin.Oniga@cs.utcluj.ro (F. Oniga).

https://doi.org/10.1016/j.imavis.2022.104613

size and the inference time, while achieving improved detection accuracy.

Our main contributions are:

- To improve detection results, we integrate Mish [16] as an activation function and we fine-tune the data augmentation parameters
- We propose a module for cross-stage spatial pyramid pooling named Mini Cross-Stage Partial Spatial Pyramid Pooling (MCSP-SPP), designed for mobile devices. This module is faster and smaller than Spatial Pyramid Pooling (SPP) [17], with a similar complexity to Cross-Stage Partial Spatial Pyramid Pooling (CSP-SPP) [13] but with improved detection quality
- With a proposed custom strategy, we selectively reduce the number of convolution filters in the last convolutional layers by 50%, with minimal performance impact. Additionally, we reduce the input size to further increase real-time performance.

This paper is organized as follows. Related work on object detection and network compression are introduced in Section 2. The proposed architecture Mini-YOLOv4-tiny is introduced and presented in Section 3. In Section 4, we present and discuss experimental results and ablation studies, with a focus on model size, Billion Floating Point Operations per second (BFLOP/s), mean Average Precision (mAP) and the real-time performance measured in Frames per Second (FPS). In addition, we compare results obtained by the Mini-YOLOv4-tiny with other SOTA lightweight models and perform a qualitative comparison between YOLOv4-tiny and our proposed Mini-YOLOv4-tiny. Finally, we draw conclusions in Section 5.

2. Related work

In the first subsection, we discuss the object detection task and some approaches from literature, both classical image processing and deep learning techniques. Deep learning techniques include both anchor-based detectors and anchor-free detectors. For anchor-based detectors, we consider single-stage detectors and two-stage detectors. In the second subsection, we present the current state of lightweight object detectors.

2.1. Object detectors

In this context, object detection aims to identify the object category and its localization in pictures. Classical image processing techniques used for object detection, such as histograms of oriented gradients [1], the deformable part-based model proposed in [2] and the Viola Jones detectors [18] use sliding windows and manually engineered feature extraction methods. However, their detection performance on large datasets and real-time performance make them unsuitable for low computing devices. In 2012, AlexNet [3] achieved SOTA results on ImageNet [4], and kick-started the popularization of Convolutional Neural Networks (CNNs). CNNs learn from data what features to use in order to achieve high performance in the field of object detection.

Existing object detectors can be split into two groups: anchor-based detectors and anchor-free detectors. Anchor-based detectors can be divided into two-stage detectors and single-stage detectors. The typical workflow for two-stage detectors [5,6,15,17,19,20], consists of (1) generating region proposals from the image and (2) generating the objects' bounding boxes and classifications. Two-stage detectors are difficult to deploy on typical CPU / ARM devices, however they tend to have more accurate object detection.

Single-stage detectors [8–13] usually find a better balance between speed and accuracy. YOLOv1 [8] was the first publication where the object detection problem was formulated as a regression and when compared to two-stage detectors, it was much faster. However, it performed worse in localizing and detecting small objects. SSD [9] is able to detect objects at multiple scales (by discretizing the output space of bounding boxes into a set of default boxes over different aspect ratios and scales), is more easily deployed on mobile devices, yet its accuracy can be improved. RetinaNet [21] introduces focal loss to solve the class imbalance problem, enabling higher accuracy than some two-stage detectors. YOLOv2 [10] improves upon YOLOv1 through the use of batch normalization, multiscale training methods and by using Darknet-19 backbone for feature extraction. YOLOv3 [11] improves on YOLOv2 by expanding to Darknet-53, using multiscale feature fusion and adding residual modules, achieving higher accuracy and real-time performance on powerful Graphics Processing Units (GPUs). YOLOv4 [12] improves on YOLOv3 through weighted residuals links, data augmentation and DropBlock regularization, while having higher processing speed and accuracy. However, on edge devices with limited storage and computational power, the YOLO series cannot meet real-time requirements. YOLOv4-tiny [13] is one of the latest improvements in the YOLO series for lightweight models. It has significantly fewer model parameters, making it suitable for deployment on mobile devices. However, the number of calculations is still large and there is still room for improvement in terms of inference speed and accuracy of the model.

Anchor-free detectors [22–24] do not use anchor boxes. In YOLOv1 [8] the image is divided into multiple grids and bounding boxes are predicted at the point near the center of each object. In CornerNet [22], pairs of corners of bounding boxes are detected without designing prior boxes for the anchor boxes. CenterNet [23] eliminates the need for determining the upper left and bottom right corners, directly identifying the center point of the bounding box. In FCOS [24], the authors formulate object detection in a per-pixel prediction method and propose the "centerness" branch. The anchor free detectors solve some of the problems of anchor-based detectors, while reducing the memory cost. Generalized Focal Loss (GFL) [25] eliminates the "centerness" branch in FCOS and merges the quality estimation into the class prediction vector to generate a joint representation between the classification and localization quality. ATSS [26] proposes an adaptive training sample selection for automatically selecting positive and negative samples according to the statistical characteristics of objects.

2.2. Lighweight object detectors

Lightweight object detectors are designed for mobile devices and other resource-constrained devices. YOLO-Lite [27] is one of the first lightweight networks designed for Non-GPU computers. SlimYOLOv3 [28] performs channel pruning on convolutional layers, starting from YOLOv3, and needs fewer parameters than YOLOv3-tiny, however it is only tested on a drone dataset. Tinier-YOLO [29] improves YOLOv3-tiny by integrating the fire module of SqueezeNet [30] into its architecture, connecting the fire modules through dense connections. They have lower computational costs and achieve better results on the Pascal VOC and COCO datasets. YOLO Nano [31] outperforms YOLOv2-tiny and YOLOv3-tiny on Pascal VOC dataset [32], while using a smaller model (only 4 MB) with fewer FLOPs. The main components of YOLO Nano are a fully-connected attention (FCA) module, an Expansion-Projection (EP) module and a residual Projection-Expansion-Projection (PEP) module. LittleYOLO-SPP [33] incorporates spatial pyramid pooling (SPP) into YOLOV3-tiny and uses Generalized Intersection Over Union (GIoU) [34] for the bounding box regression. However, it only tests the proposed method for three classes in the COCO dataset and the source code is not publicly available. Lightweight SSD [35] proposes an improved lightweight backbone network and an efficient feature fusion network for the detection neck. It also introduces two bag of freebies methods which enables it to achieve competitive accuracy with a lower computational cost on mobile devices.

Jiang et al. [36] are among the first to work on optimizing the YOLOv4-tiny, proposing to substitute the CSPBlock with a ResBlock-D module. This enables them to increase the detection speed, but also reduces the accuracy, which they compensate for by using an

auxiliary residual network block (which uses CBAM [37]). YOLObile [38] has fewer FLOPs and parameters than YOLOv4-tiny, while achieving better AP and FPS by using a block-punched pruning scheme. However, their method requires a specific compiler to achieve top speed, which restricts its availability and application. In [39], the authors propose using a twin head architecture for the YOLO layers, while replacing all max pooling layers with convolutional ones and integrating ECA-Net [40] and Mish activation [16]. They achieve better results than YOLOv4-tiny with fewer parameters but higher FLOPs. In [41], the authors propose a lightweight Raw Feature Collection and Redistribution (RFCR) module that combines multi-scale features and is compatible with various backbones and detection heads. They insert it into YOLOV4-tiny and increase the Average Precision (AP) with a small drop in the FPS. TRC-YOLO [42] reduces the number of convolution cores of YOLOv4-tiny and introduces an hourglass Cross Stage Partial ResNet (CSPResNet). Then, they add a Receiver Field Block (RFB) to the backbone to increase the receptive field and apply channel and spatial attention to the feature extraction, enabling them to reduce the model size and improve performance. Trident-YOLO [43] introduces a Cross-Stage Partial Receiver Field Block (CSP-RFB), which is lightweight and enlarges the receptive field of the model with lower computational complexity. They introduce the Trident Feature Pyramid Network (Trident-FPN) to merge features from different backbone levels and CSP-SPPs, which is designed for mobile devices and obtains richer features.

3. Proposed solution

Based on YOLOv4-tiny (Fig. 1 a), we have developed a new lightweight object detector named Mini-YOLOv4-tiny (Fig. 1 b). We selectively reduce the number of convolution filters in the last convolutional layers by 50%, (optionally) downscale the input size to 288 pixels, replace Leaky-RELU activation with Mish, fine-tune the data augmentation parameters, and add our proposed MCSP-SPP module. In this section, we will discuss the detector's architecture and the proposed improvements.

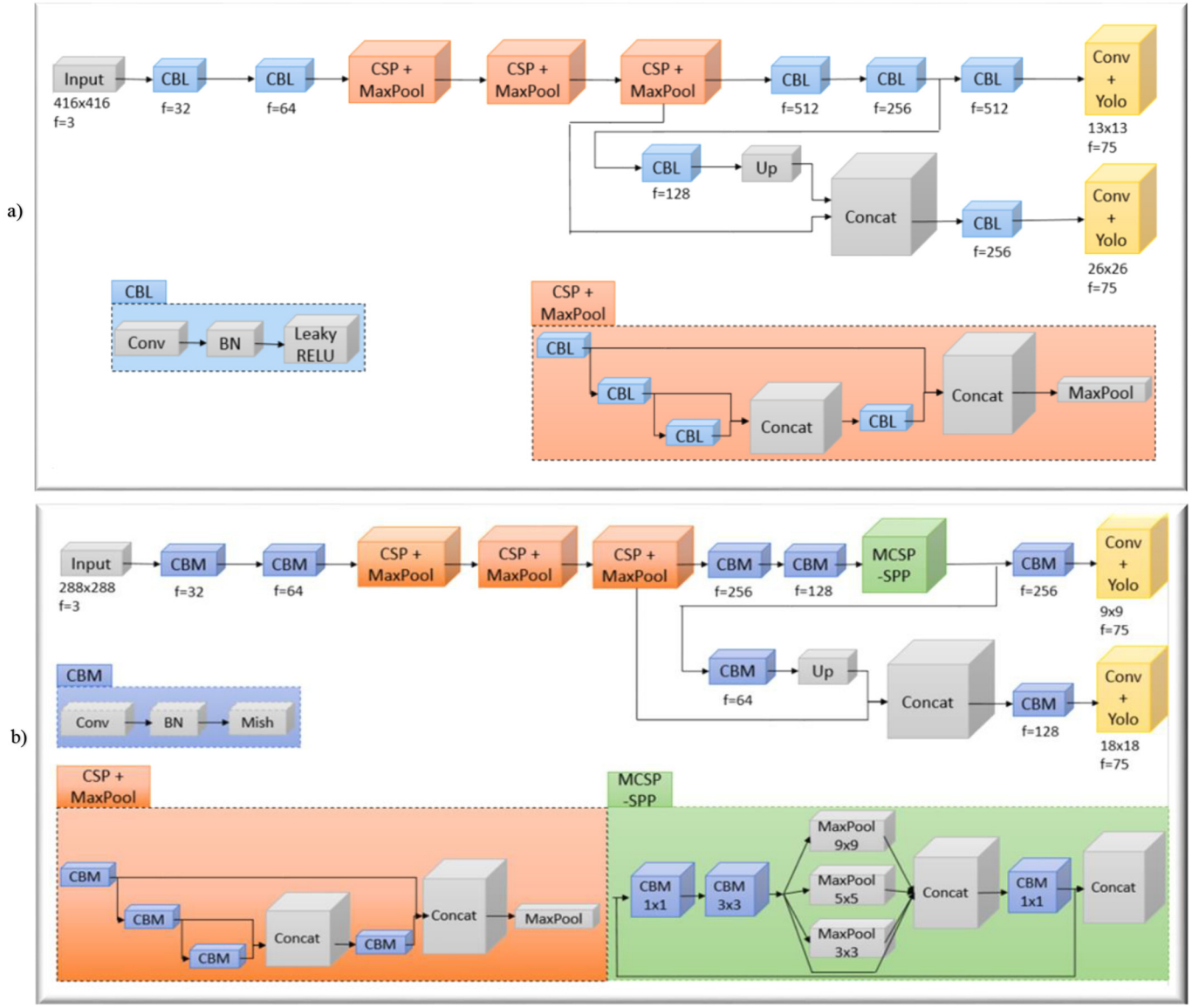

Fig. 1. a) YOLOv4-tiny original architecture: the backbone is made of CBL blocks and CSP + MaxPool blocks, the detection head is mainly composed of CBL blocks and two Yolo heads; b) Proposed Mini-YOLOv4-tiny architecture: all CBL blocks are replaced with CBM blocks, MCSP-SPP is added before the last CBM block and the number of convolutional filters in the last 3 CBM blocks is reduced by half.

3.1. Motivation

In Fig. 1 a), we present the network structure of YOLOv4-tiny. The principal module in the backbone is the Cross-stage Partial (CSP), which consists of a Convolutional layer followed by Batch normalization and the Leaky-RELU activation function (CBL). The module also features skip connections. To achieve a good balance between detection efficiency and accuracy, YOLOv4-tiny uses three of these modules in the backbone to gradually extract the image features. For the detection head, YOLOv4-tiny uses two heads. The detection section consists of one CBL and convolutional layer, followed by the detection layer. The backbone's feature maps are upsampled and combined with the residual connection from the same feature map resolution for the detection head at the second scale. The first scale is for larger objects, with a feature maps size of 13×13, while the second scale is for smaller objects, with a feature maps size of 26×26.

However, some convolution layers in the backbone and in the detection head use 512 and 256 convolution filters, which leads to a large number of model parameters and reduces the inference speed of the model. Additionally, not all of the 512/256 filter parameters have significant contributions to model predictions.

To address the limitations of YOLOv4-tiny, we propose Mini-YOLOv4-tiny. The purpose of Mini-YOLOv4-tiny is to obtain a smaller, faster and better performing model that can run on a resource constrained device.

3.2. Proposed structure of Mini-YOLOv4-tiny

Fig. 1 b) illustrates the structure of the proposed Mini-YOLOv4-tiny. First, we propose a bottom-up strategy to eliminate the convolutional filters in the model architecture with minimal impact on the accuracy of the model and then reduce the input size of the network. The model compression in the first step significantly reduces the size and the number of FLOPS, but also negatively impacts the performance of the model. In the next steps, we focus on methods to increase performance with few additional computations. First, we switch the activation function from Leaky RELU to Mish (CBL are replaced by CBM) and fine-tune the data augmentation parameters. Next, we propose MCSP-SPP, a version of CSP-SPP adapted for low computation, which allows the model to integrate deep multi-scale features in the network and increase the receptive field. These design choices were guided and justified by the experimental results and ablation studies presented in section 4.

3.3. Model scaling

Wang et al. [13] suggest that model scaling methods should mainly address the image size, the number of layers and the number of channels. Down scaling the image size can boost the real-time performance of the model but has a negative impact on the detection quality. Typically, a larger image size is associated with higher performance of the model, especially for smaller scale objects. Blindly reducing the number of layers can greatly decrease the performance, due to reduced model capacity and limited representativity of the features. Modifying the number of channels of the convolution layers can significantly reduce the model size. This can have a small impact on the performance of the model if the channel drop begins from the bottom layers up. These bottom layers contain higher level features, while their drop can be compensated by the remaining channels. This is the approach that we are following in this work. We start by eliminating half of the convolution filters starting from the last convolutional layer, incrementally moving up to the next convolutional layer until the first convolutional layer from the backbone is reached.

3.4. Mish

In the experiments performed in [13], Mish outperforms Leaky RELU in Scaled-YOLOv4 on the MS COCO dataset. However, all the activation functions in YOLOv4-tiny are Leaky RELU. Thus, in Mini-YOLOv4-tiny we replace Leaky RELU with Mish to improve detections results.

3.5. Data augmentation

The aim of data augmentation is to increase the robustness of the model for images coming from different environments, mainly through pixel-wise adjustments such as photometric distortions and geometric distortions. Photometric distortions are often addressed through random adjustments of the brightness, contrast, hue, saturation and the level of noise in an image. To handle geometric distortions, random scaling, cropping, flipping and rotating are commonly used. These data augmentation techniques are generally very effective for image classification and object detection. We mainly fine-tune the photometric distortions strength on the hue, saturation and value, which leads to improved performance with no additional computation required for model inference.

3.6. MCSP-SPP

SPP was first integrated into a CNN in the form of spatial pyramid matching (SPM) in [17], where the authors use a max-pool layer instead of a bag-of-words layer. In YOLOv3 [11] SPP-YOLO was introduced, which enabled an increase of AP50 by 2.7% on the MS COCO dataset, while requiring only 0.5% additional computations. YOLOv4 [12] also uses the SPP-YOLO module and the model performance surpasses that of YOLOv3. Scaled-YOLOv4 [13] takes it to the next level with CSP-SPP, which uses a cross-stage process to perform the down sampling convolution operations.

The SPP structures mentioned above are effective for improving the model performance, but they are designed for object detection models with a large number of parameters and are not suitable for lightweight object detectors. Wang et al. [43] propose CSP-SPPs, where they delete one 1×1 and one 3×3 convolutional layers from CSP-SPP [13] and switch to fusing the output of the three different scales of max-pool layers in a shortcut manner instead of concatenation.

We also start from CSP-SPP [13] and propose our MCSP-SPP (see Fig. 2). First, we eliminate two 1×1 and one 3×3 convolutional layers. In contrast to CSP-SPPs [43], we switch all activation functions from Leaky-RELU to Mish and then downscale the kernel size for each of the three max-pool layers to 9×9, 5×5 and 3×3 and concatenate the output (instead of fusing) from the max-pool layers so a richer representation of the feature maps is obtained. We prove the effectiveness of our proposed design through the ablation experiment in Section 4.

4. Experimental results

In this section, we present the experimental settings and datasets. Then, the performance comparison under the metrics of mAP, BFLOP/s, model size and FPS are carried out. We conduct ablation studies on the Pascal VOC dataset for model scaling, switching the activation function, fine-tuning the data augmentation parameters and introducing various variants of SPP, including the proposed MCSP-SPP. Finally, we do a qualitative performance comparison between the predictions of YOLOv4-tiny and our proposed Mini-YOLOv4-tiny.

4.1. Experimental settings

The framework used for training is Darknet, which is an open-source neural network framework written in C and CUDA, which supports multiple CPUs and GPUs. The optimizer we use is the Stochastic

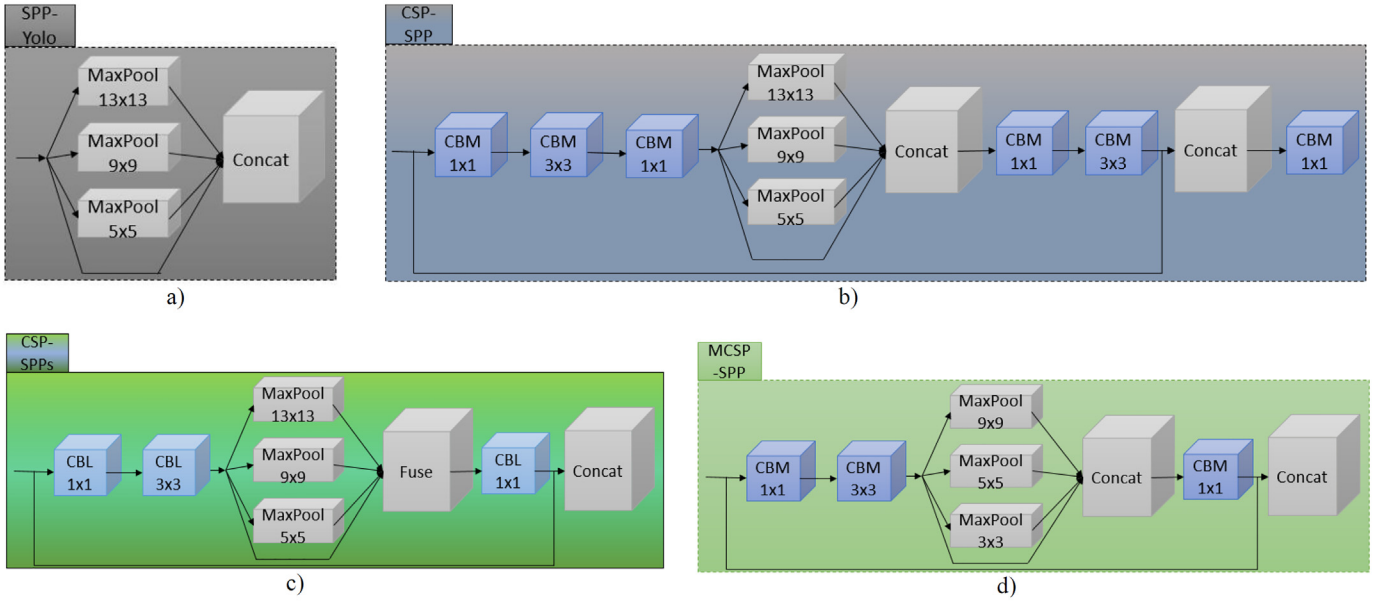

Fig. 2. Four different SPP structures: (a) SPP structure in YOLO v3 [11] and YOLO v4 [12], which fuses the input feature map with the output feature map to obtain a more comprehensive feature; (b) CSP-SPP structure in scaled YOLO v4 [13], which achieves higher performance with fewer parameters and lower computational complexity; (c) CSP-SPPs [43], which is derived from CSP-SPP, where the authors use CBL blocks instead of CBM, and drop 2 of these CBM blocks from CSP-SPP, while fusing the output of the max pooling operations; d) Our proposed MCSP-SPP, which uses only 3 CBM blocks, downscales the kernel size for the max-pool operations, and finally concatenates the output from the max-pool operations.

Gradient Descent (SGD) with momentum 0.9 and an initial learning rate of 0.00261. We set the batch size to 64 with 2 subdivisions. The model iterates 42,000 times on the PASCAL VOC dataset and 500,500 times on the MS COCO dataset. The YOLO layer parameter settings are consistent with those of YOLOV4-tiny. We use the pre-trained weights YOLOv4-tiny.conv.29. We train the networks on a single NVidia V100. The detection speed is tested on NVidia Jetson Nano and Raspberry Pi 4 model B. In Table 1 we show the configuration comparison between Jetson Nano and Raspberry Pi 4 model B.

4.2. Experimental datasets

The datasets used in this paper are PASCAL VOC [32] and MS COCO [44], typically used for image classification, object detection and image segmentation tasks.

In the PASCAL VOC dataset, we include both PASCAL VOC 2007 and PASCAL VOC 2012, which has 20 object classes and a total of 21,503 images. When calculating the AP and mAP values, we use the default value of 0.5 for the intersection over union (IOU) threshold. The COCO dataset contains 80 object categories, with 122,264 training and validation images. The mAP value on this dataset is the average of 10 values of IOU from 0.5 to 0.95 and mAP is compared for small, medium and large objects. Table 2 shows the comparison between the two datasets.

Table 1
Hardware configuration of NVIDIA Jetson Nano and Raspberry PI 4 Model B.

	NVIDIA Jetson Nano	Raspberry PI 4 Model B
Accelerator	128-core NVIDIA Maxwell GPU	Broadcom VideoCore VI
TFLOPs (FP16)	0.5 TFLOP	0.02 TFLOP
CPU	Quad-core ARM Cortex-A57, 1.43 GHz	Quad-core ARM Cortex-A72, 1.5 GHz
Memory Size	2 GB	4 GB
Memory Type	LPDDR4	LPDDR4
Interface	PCIe	USB 3.0
Power	5 W–10 W	3 W–6.25 W
Price	61$	35-49

Table 2
PASCAL VOC and MS COCO datasets details.

	Pascal VOC	MS COCO
Number of classes	20	80
Samples in training set	16,551	117,264
Samples in validation set	4,952	5,000
Number of ground-truth boxes	52,090	866,678
Average number of boxes per image	2.1	7.4

4.3. Ablation experiment on model scaling

To better understand the impact of model scaling through cutting the number of convolution filters, we conduct an ablation study and present the achieved results in Table 3. In the first experiment, we reduce the number of convolution filters by half for all CBL blocks and then train a model. Next, starting from the YOLOv4-tiny original architecture, we eliminate half of the convolution filters starting from the last CBL blocks before the YOLO heads and then move upwards towards the next CBL blocks until the last CSP + Maxpool block in the backbone. In the last step, we reduce the input size from 416×416 to 288×288. It can be observed that reducing all convolution filters by half reduces the model size by 74.7%, the BFLOPs by 74.6%, but also reduces the average IoU by 6.61% and the mAP by 29.62%. Finally, when the last three CBLs are halved, the model size can be reduced by 46.1% (from 23.14 MB),

Table 3
Ablation study on model scaling on VOC.

Model	Model size (MB)	BFLOPs	Average IoU	mAP (%)
YOLOv4-tiny	23.14	6.817	53.26	**71.77**
YOLOv4-tiny, half filters for **all CBLs**	5.86	1.733	46.65	42.15
YOLOv4-tiny, half filters for **last CBL**	18.99	6.000	52.95	71.38
YOLOv4-tiny, half filters for **last 2 CBLs**	17.20	5.770	52.33	70.57
YOLOv4-tiny, half filters for **last 3 CBLs**	12.46	5.360	52.12	69.66
YOLOv4-tiny, half filters for last 3 CBLs, **input size 288 × 288**	12.46	2.569	54.06	69.51

BFLOPs by 21.3% with an average IoU decrease of 1.14% and mAP is decreased by only 2.11%. To further reduce the required computations, we reduce the input size from the original 416 × 416 resolution of YOLOv4-tiny to 288 × 288, which has no impact on the model size in MB, but significantly reduces the BFLOPs by 52% (from 5.360), increases the average IoU by 1.94% and slightly decreases the mAP by 0.15%. The increase in average IoU might be due to the regularization effect on this dataset after the input size downscaling, but the mAP still decreases.

4.4. Ablation experiment on mish activation and data augmentations

Replacing the Leaky-RELU activation function with Mish and then fine-tuning the data augmentation parameters of the model leads to an improved performance as can be observed in Table 4. Switching to the Mish activation function preserves model size and BFLOPs, while increasing the IoU by 0.57% (from 54.06%) and the mAP by 0.35%. Fine-tuning the data augmentation increases the IoU by 0.25% and mAP by 0.31%.

4.5. Ablation experiment on SPP

In Table 5, we present the comparison between the performance of the model after integrating different SPP structures. SPP-YOLO [11] is designed for a large model so its BFLOP and the model size are the largest. CSP-SPP [14] effectively reduces the number of model parameters and the model size, while increasing the mAP of the model. CSP-SSPs [43] further decreases the model size and BFLOPS, while increasing the IoU and mAP. Our proposed MCSP-SPP slightly increases the model size by 16.7% (compared to the model without any SPP) and the BFLOPs by 3.4%, while increasing the IoU by 2.12% and the mAP by 1.9%.

4.6. Ablation experiment mini-YOLOV4-tiny (288) on VOC

To illustrate the effectiveness of the Mini-YOLOv4-tiny and the influence of each proposed improvement, we first perform model scaling and then gradually add Mish, fine-tune the data augmentation and add MCSP-SPP. Table 6 shows the ablation experiment in terms of mAP, model size and BFLOPs. It can be observed that the model with all components included is only 62.8% of the baseline YOLOV4-tiny model size, BFLOPs are only 38.9% of the baseline, while the IoU increases by 3.73% and mAP increases by 0.3%.

Table 4
Ablation study on activation function and data augmentation on VOC.

Model	Model size (MB)	BFLOPs	Average IoU	mAP (%)
YOLOv4-tiny, half filters for last 3 CBLs, input size 288 × 288, **Leaky RELU**	12.46	2.569	54.06	69.51
YOLOv4-tiny, half filters for last 3 CBLs, input size 288 × 288, **Mish**	12.46	2.569	54.63	69.86
YOLOv4-tiny, half filters for last 3 CBLs, input size 288 × 288, **Mish, data augmentation**	**12.46**	**2.569**	**54.88**	**70.17**

4.7. Comparison on real time performance

We used the official Darknet framework "detector demo" function with the flags "-benchmark -dont_show" when running the inference time evaluation. We evaluate on both NVidia Jetson Nano and Raspberry Pi 4 model B, and show the results in Table 7. Model scaling leads to the best real-time performance but compromises the mAP to some extent. When we only use the model scaling, it achieves 200% higher FPS than the baseline on Jetson Nano and 260% higher FPS on Raspberry Pi, while requiring 62% fewer BFLOPs and having a 46% smaller model size. After adding all the proposed improvements, the processing time does not increase significantly. The proposed approach (288 input size) achieves 185% higher FPS on Jetson Nano, 236% higher FPS on Raspberry PI, while requiring 61% BFLOPs and having a 37% smaller model size.

4.8. Performance comparison between Mini-YOLOv4-tiny and other lightweight object detection algorithms on Pascal VOC

To compare the performance of Mini-YOLOv4-tiny and other lightweight object detectors on the Pascal VOC dataset, we select YOLOv2-tiny [10], YOLOv3-tiny [11], YOLOv4-tiny [13], SquezzeNet SSD [9], MobileNet SSD [9], YOLO Nano [31],Trident YOLO [43], and Rescaled YOLOv4-tiny [45] and present the comparisons in Table 8. The smallest model size is obtained by YOLO Nano (4.0 MB), but it is outperformed on the mAP by the other detectors. Our proposed model outperforms the model with the highest mAP (YOLOv4-tiny, 71.77%) by 0.3%, while having 62.8% of its size and requiring only 38.9% of its BFLOPS. Mini-YOLOv4-tiny also has the fewest BFLOPs (2.657) of the compared models.

4.9. Performance of Mini-YOLOv4-tiny on MS-COCO

To further verify the performance of the Mini-YOLOv4-tiny model, we evaluate it on the MS COCO dataset. We train the model with three different input sizes, 288, 320 and 416.

We first compare the performance with YOLOv4-tiny and show the results in Table 9. Mini-YOLOv4-tiny with input size 288 needs 37.4% less parameters and 61.2% less BFLOPs, while IoU increases by 2.76%, AP (0.50:0.95) decreases by 1.4%, AP (0.5) decreases by 3%, and AP (0.75) decreases by 1%. The model performs slightly worse on small boxes (2.9% worse) and on medium boxes (4.6% worse), while outperforms the baseline on large boxes by 3.5%.

Mini-YOLOv4-tiny with input size 416 needs the same number of parameters as the model trained with input size 288 (3.79 M). It requires 19% fewer BFLOPs than the baseline, with a 4.02% increase in IoU, 2.8% increase in AP (0.50:0.95), 2.1% increase in AP (0.5), and 2.4% increase in AP (0.75). The model performs slightly worse on small boxes (1.1% worse), while outperforming the baseline on medium boxes (1.1% better) and large boxes (3.4% better).

4.10. Performance comparison between Mini-YOLOv4-tiny and other lightweight object detection algorithms on MS-COCO

Now we compare our proposed model with other lightweight object detection algorithms on the MS COCO dataset, as shown in Table 10. In our comparison, we include only models that have less than 10 BFLOPs,

Table 5
Ablation study results for SPP on VOC.

Model	Model size (MB)	BFLOPs	Average IoU	mAP (%)
YOLOv4-tiny, half filters - last 3 CBLs, input 288 × 288, Mish, data augmentation	**12.46**	2.569	54.88	70.17
YOLOv4-tiny, half filters - last 3 CBLs, input 288 × 288, Mish, data aug., **SPP-YOLO**	16.01	2.718	56.31	70.61
YOLOv4-tiny, half filters - last 3 CBLs, input 288 × 288, Mish, data aug., **CSP-SPP**	14.196	2.642	56.07	71.44
YOLOv4-tiny, half filters for last 3 CBLs, input 288 × 288, Mish, data aug., **CSP-SPPs**	13.787	**2.485**	56.25	71.76
YOLOv4-tiny, half filters for last 3 CBLs, input 288 × 288, Mish, data aug., **MCSP-SPP (ours)**	14.54	2.657	**57.00**	**72.07**

Table 6
Ablation experiment on the proposed Mini-YOLOV4-tiny (input size 288) on VOC.

	YOLOv4-tiny	Mini-YOLOv4-tiny (288)			
Model scaling		+	+	+	+
Mish			+	+	+
Data augm.				+	+
MCSP-SPP					+
Model size (MB)	23.14	12.46	12.46	**12.46**	14.54
BFLOP/s	6.817	2.569	2.569	**2.569**	2.657
Average IOU	53.26	54.06	54.63	54.88	**56.99**
mAP (%)	71.77	69.51	69.86	70.17	**72.07**

Table 7
Comparison of the real-time performance (input size 288) on VOC.

	YOLOv4-tiny	Mini-YOLOv4-tiny			
Model scaling		+	+	+	+
Mish			+	+	+
Data augm.				+	+
MCSP-SPP					+
Model size (MB)	23.14	12.46	12.46	12.46	14.54
BFLOP/s	6.817	2.569	2.569	2.569	2.657
FPS on NVidia Jetson Nano	19.5	39.0	37.5	37.5	36.0
FPS on Raspberry Pi 4 model B	0.28	0.73	0.67	0.67	0.66

Table 8
Performance comparison with other lightweight detectors on VOC.

Model	Model size (MB)	BFLOPs	mAP (%)
YOLOv4-tiny	23.14	6.817	71.77
YOLOv2-tiny	63.5	6.977	44.53
YOLOv3-tiny	34.9	5.477	52.78
SqueezeNet SSD	22.7	4.499	64.3
MobileNet SSD	23.3	4.288	66.1
YOLO Nano	**4.0**	4.570	69.1
Trident YOLO	10.9	5.199	67.6
Rescaled YOLOv4-tiny	30.83	5.529	66.73
Mini-YOLOv4-tiny (288, ours)	14.54	**2.657**	**72.07**

such as YOLOv3-tiny [11], YOLOv4-tiny [13], Trident YOLO [43], TRC-YOLO [42],Tinier Yolo [29], YOLO-Ret-EB3 [41], YOLOv4-tiny + RFCR [41] and Rescaled YOLOv4-tiny [45]. We show that our proposed method has the best speed-model size-accuracy trade-off. Our Mini-YOLOV4-tiny with input size 288 significantly outperforms the model with the fewest BFLOPs (Tinier-YOLO) on AP (0.5:0.95) by 3.2%, on AP (0.5) by 2.9% and on AP (0.75) by 4.3%, while having only 4.7% additional BFLOPs. The highest performing model is our Mini-YOLOV4-tiny with input size 416, which needs 3.79 M parameters and 5.59 BFLOPs (19% less than the YOLOV4-tiny), and achieves an AP (0.50:0.95) of 23.4%, AP (0.5) of 42.2% and AP (0.75) of 23.4%.

4.11. Comparison with the latest YOLO lighweight detector on MS-COCO

Recently available online (Arxiv), YOLOv7-tiny [46] is the latest member of the YOLO family of lightweight object detectors. For comparing its performance with our proposed Mini-YOLOv4-tiny, we trained YOLOv7-tiny by following the training configuration suggested by the authors. We used the same number of training iterations as stated in sub-section 4.1. In addition to the metrics used in the previous section, we evaluated the IoU metric and the FPS on Nvidia Tesla V100. We present the results in Table 11. Our Mini-YOLOv4-tiny with an input size 416 has a 40% smaller model size, 39% less model parameters, 4% less BFLOPs and achieves 204% more FPS. When it comes performance, Mini-YOLOv4-tiny achieves 0.5% higher IoU, while being outperformed by 5.3% on the AP 0.5:0.95 metric, by 1% on the AP 0.5 metric and by 8.5% on the AP 0.75 metric.

Table 10
Comparison with other lightweight detection models on the MS COCO data set.

Model	Input Size	Params (M)	BFLOPs	AP 0.5:0.95	AP 0.5	AP 0.75
YOLOv4-tiny	416	6.06	6.91	21.6	40.1	21.0
YOLOv3-tiny	416	8.86	5.62	15.3	33.1	12.4
Trident-YOLO	416	–	5.30	18.8	37.0	17.3
TRC-YOLO	416	–	3.08	18.4	38.4	15.6
Tinier-YOLO	416	–	**2.56**	17.0	34.0	15.7
YOLO-ReT-EB3	416	8.87	7.98	20.8	39.1	22.6
YOLO-ReT-EB3	320	8.87	6.14	19.4	36.5	19.3
YOLOv4-tiny + RFCR	416	–	7.075	22.9	41.5	23.3
Rescaled YOLOv4-tiny	416	7.77	5.529	23.2	**42.7**	23.1
Mini-YOLOv4-tiny	416	**3.79**	5.59	**23.4**	42.2	**23.4**
Mini-YOLOv4-tiny	320	3.79	3.30	21.3	38.8	21.3
Mini-YOLOv4-tiny	288	3.79	2.68	20.2	36.9	20.0

4.12. Qualitative performance comparison between YOLOv4-tiny and Mini-YOLOv4-tiny on MS-COCO

In Fig. 3, we present some scenarios where our proposed approach performs better than the baseline. In these specific examples, YOLOv4-tiny has false detections (dog instead of bear, and bed instead of sofa) and assigns lower scores to large bounding boxes. Mini-YOLOv4-tiny has fewer false detections and assigns higher scores to the correct object categories.

5. Conclusions

This paper presents a neural network architecture for object detection with improved real-time performance, suitable for devices with limited computing power, called Mini-YOLOV4-Tiny. We compress and speed up the model by selectively cutting the width of the last convolutional layers. Then, we increase model performance with minimal impact on memory and inference time. We replace the Leaky-RELU activation function with Mish and fine-tune the data augmentation parameters. After introducing our proposed MCSP-SPP module, the Mini-YOLOV4-tiny series achieves state-of-the-art results among lightweight detectors on VOC and COCO datasets.

Table 9
Performance of YOLOv4-tiny and Mini-YOLOv4-tiny on the MS COCO data set.

Model	Input Size	Params (M)	BFLOPs	Average IoU	AP 0.5:0.95	AP 0.50	AP 0.75	AP S 0.5:0.95	AP M 0.5:0.95	AP L 0.5:0.95	FPS Nano
YOLOv4-tiny	416	6.06	6.91	49.1	21.6	40.1	21.0	**10.2**	25.7	30.2	18.7
Mini-YOLOv4-tiny (ours)	416	3.79 (−37%)	5.59 (−19%)	**53.12 (+4)**	**23.4 (+2.8)**	**42.2 (+2.1)**	**23.4 (+2.4)**	9.1 (−1.1)	**26.8 (+1.1)**	33.6 (+3.4)	21.3 (+14%)
Mini-YOLOv4-tiny (ours)	320	3.79 (−37%)	3.30 (−52%)	52.75 (+3.7)	21.3 (−0.3)	38.8 (−1.3)	21.4 (+0.4)	8.2 (−2)	23.1 (−2.6)	**34.7 (+4.5)**	32.2 (+72%)
Mini-YOLOv4-tiny (ours)	288	3.79 (−37%)	**2.68 (−61%)**	51.86 (+2.8)	20.2 (−1.4)	37.1 (−3)	20.0 (−1)	7.3 (−2.9)	21.3 (−4.3)	33.7 (+3.5)	**34.7 (+86%)**

Table 11
Performance comparison between YOLOv7-tiny and Mini-YOLOv4-tiny on MS COCO data set.

Model	Input Size	Model Size (MB)	Params (M)	BFLOPs	IoU	AP 0.5:0.95	AP 0.50	AP 0.75	FPS V100
YOLOv7-tiny	416	24.83	6.2	5.8	52.62	**28.7**	**43.2**	**31.9**	218
Mini-YOLOv4-tiny (ours)	416	**14.81** (−40%)	**3.79** (−39%)	5.59 (−3.6%)	**53.12** (+0.5)	23.4 (−5.3)	42.2 (−1.0)	23.4 (−8.5)	444.1 (+204%)
Mini-YOLOv4-tiny (ours)	320	**14.81** (−40%)	**3.79** (−39%)	3.30 (−43%)	52.75 (+0.1)	21.3 (−7.4)	38.8 (−4.4)	21.4 (−10.5)	470.3 (+216%)
Mini-YOLOv4-tiny (ours)	288	**14.81** (−40%)	**3.79** (−39%)	**2.68** (−54%)	51.86 (−0.8)	20.2 (−8.5)	37.1 (−6.1)	20.0 (−11.9)	**507.1** (+233%)

Fig. 3. Comparison of false detections and detection scores. The YOLOv4-tiny predictions are on the left column and the Mini-YOLOv4-tiny predictions are on the right column. In all 3 cases, our method improves the detection results. In group a), YOLOv4-tiny incorrectly predicts the bear to be a dog, while our method correctly identifies the bear and has a high score. As for groups b) and c), our method assigns higher detection scores and tighter object localizations, and correctly classifies the object in group b) as a sofa instead of a bed.

CRediT authorship contribution statement

Dan-Sebastian Bacea: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Writing – original draft, Visualization, Software, Data curation. **Florin Oniga:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision.

Data availability

No data was used for the research described in the article.

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.

Acknowledgement

This work was supported by the "DeepPerception – Deep Learning Based 3D Perception for Autonomous Driving" grant funded by Romanian Ministry of Education and Research, code PN-III-P4-PCE-

2021-1134. Experiments were done on the computing infrastructure of the CLOUDUT Project, cofunded by the European Fund of Regional Development through the Competitiveness Operational Programme 2014-2020, contract no. 235/2020.

References

[1] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 2005.
[2] P.F. Felzenswalb, R.B. Girshick, D. McAllester, D. Ramanan, Object detection with discriminatively trained part-based models, IEEE Trans. Pattern Anal. Mach. Intell. 32 (9) (2010) 1627–1645.
[3] A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems, Lake Tahoe, Nevada, United States, 2012.
[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.
[5] R. Girshick, Fast r-cnn, Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 2015.
[6] S. Ren, H. Kaiming, G. Ross, S. Jian, Faster r-cnn: Towards real-time object detection with region proposal networks, Advances in Neural Information Processing Systems 28, Montreal, Canada, 2015.
[7] J. Dai, Y. Li, K. He, J. Sun, R-FCN: object detection via region-based fully convolutional networks, Advances in Neural Information Processing Systems 29, Barcelona, Spain, 2016.
[8] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, Nevada, USA, 2016.
[9] W. Liu, D. Anguelov, E. Dumitru, S. Christian, R. Scott, F. Cheng-Yang, A.C. Berg, Ssd: single shot multibox detector, European Conference on Computer Vision, Amsterdam, Netherlands, 2016.
[10] J. Redmon, A. Farhadi, YOLO9000: better, faster, stronger, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, Hi, USA, 2017.
[11] J. Redmon, A. Farhadi, Yolov3: An incremental improvement, arXiv preprint arXiv: 1804.02767 2018.
[12] A. Bochkovskiy, C.-Y. Wang, H.-Y.M. Liao, Yolov4: Optimal speed and accuracy of object detection, arXiv preprint arXiv:2004.10934 2020.
[13] C.-Y. Wang, A. Bochkovskiy, H.-Y.M. Liao, Scaled-yolov4: scaling cross stage partial network, Proceedings of the IEEE/cvf Conference on Computer Vision and Pattern Recognition, 2021.
[14] M. Tan, R. Pang, Q.V. Le, Efficientdet: scalable and efficient object detection, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seatle, WA, USA, 2020.
[15] K.G.G. He, P. Dollar, R. Girshick, Mask r-cnn, Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 2017.
[16] D. Misra, Mish: A self regularized non-monotonic activation function, arXiv preprint arXiv:1908.08681 2019.
[17] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, IEEE Trans. Pattern Anal. Mach. Intell. 37 (9) (2015) 1904–1916.
[18] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai, HI, USA, 2001.
[19] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, S. Belongie, Feature pyramid networks for object detection, IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 2017.
[20] Z. Cai, N. Vasconcelos, Cascade r-cnn: delving into high quality object detection, IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018.
[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollar, Focal loss for dense object detection, IEEE International Conference on Computer Vision, 2017.
[22] H. Law, J. Deng, Cornernet: detecting objects as paired keypoints, Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 2018.
[23] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, Q. Tian, CenterNet: keypoint triplets for object detection, Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 2019.
[24] Z. Tian, C. Shen, H. Chen, T. He, FCOS: fully convolutional one-stage object detection, Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea, 2019.
[25] X. Li, W. Wang, L. Wu, S. Chen, X. Hu, J. Li, J. Tang, J. Yang, Generalized focal loss: learning qualified and distributed bounding boxes for dense object detection, Advances in Neural Information Processing Systems, 33, , 2020.
[26] S. Zhang, C. Chi, Y. Yao, Z. Lei, S.Z. Li, Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
[27] 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 (Big Data), Seattle, WA, USA, 2018.
[28] P. Zhang, Y. Zhong, X. Li, SlimYOLOv3: narrower, faster and better for real-time UAV applications, Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea, 2019.
[29] W. Fang, L. Wang, P. Ren, Tinier-YOLO: a real-time object detection method for constrained environments, IEEE Access 8, 2019.
[30] F.N. Iandola, S.H. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutze, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, arXiv preprint arXiv:1602.07360 2016.
[31] A. Wong, M. Famouri, M.J. Shafiee, F. Li, B. Chwyl, J. Chung, YOLO nano: a highly compact you only look once convolutional neural network for object detection, 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS), Vancouver, BC, Canada, 2019.
[32] M. Everingham, L. Van Gool, C.K.I. Williams, J. Winn, A. Zisserman, The PASCAL visual object classes (VOC) challenge, Int. J. Comput. Vis. 88 (2) (2010) 303–338.
[33] E. Rani, J. Sri, LittleYOLO-SPP: a delicate real-time vehicle detection algorithm, Optik - International Journal for Light and Electron Optics, 225, , 2021.
[34] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, D. Ren, Distance-IoU loss: faster and better learning for bounding box regression, Proc. AAAI Conf. Artif. Intell. 34 (07) (2020) 12993–13000.
[35] S. Guo, Y. Liu, Y. Ni, W. Ni, Lightweight SSD: Real-Time Lightweight Single Shot Detector for Mobile Devices, VISIGRAPP, 2021.
[36] Z. Jiang, L. Zhao, S. Li, Y. Jia, Real-time object detection method based on improved YOLOv4-tiny, arXiv preprint arXiv:2011.04244 2020.
[37] S. Woo, J. Park, J.-Y. Lee, I.S. Kweon, CBAM: convolutional block attention module, Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 2018.
[38] Y. Cai, H. Li, G. Yuan, W. Niu, Y. Li, X. Tang, R. Bin, Y. Wang, YOLObile: real-time object detection on mobile devices via compression-compilation co-design, Proceedings of the AAAI Conference on Artificial Intelligence, 2021.
[39] Y. Liu, C.-W. Ma, Improving tiny YOLO with fewer model parameters, IEEE Seventh International Conference on Multimedia Big Data (BigMM), Taichung, Taiwan, 2021.
[40] Q. Wang, B. Wu, P. Zhu, P. Li, Q. Hu, ECA-Net: efficient channel attention for deep convolutional neural networks, IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.
[41] P. Ganesh, Y. Chen, Y. Yang, D. Chen, M. Winslett, YOLO-ReT: towards high accuracy real-time object detection on edge GPUs, Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2022.
[42] G. Wang, H. Ding, Z. Yang, B. Li, Y. Wang, L. Bao, TRC-YOLO: a real-time detection method for lightweight targets based on mobile devices, IET Comput. Vis. 16 (2) (2022) 126–142.
[43] G. Wang, H. Ding, B. Li, R.N. Nie, Y. Zhao, Trident-YOLO: improving the precision and speed of mobile device object detection, IET Image Process. 16 (1) (2022) 145–157.
[44] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, C.L. Zitnick, Microsoft COCO: common objects in context, European Conference on Computer Vision, Zurich, Switzerland, 2014.
[45] L. Zhao, Q. Zhang, B. Peng, L. Yang, Real-time object detector for low-end devices, J. Electron. Imaging 31 (1) (2022), 013016.
[46] W. Chien-Yao, B. Alexey, L. Hong-Yuan Mark, YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, arXiv preprint arXiv:2207.02696 2022.