https://ieeexplore.ieee.org/document/8941141 June 18, 2026

IEEE.org IEEE Xplore IEEE SA IEEE Spectrum More Sites Subscribe Donate Cart Create Account Personal Sign In

IEEE Xplore® Browse My Settings Help Institutional Sign In IEEE

All

ADVANCED SEARCH

Journals & Magazines > IEEE Access > Volume: 8

Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments

Publisher: IEEE

Cite This PDF

Wei Fang ; Lin Wang ; Peiming Ren All Authors

285 Cites in Papers

30168 Full Text Views

Open Access Comment(s)

Under a Creative Commons License

Abstract

Document Sections

I. Introduction

II. Related Work

III. The Proposed Tinier-YOLO for Constrained Environments

IV. Experimental Results

V. Conclusion

Authors

Figures

References

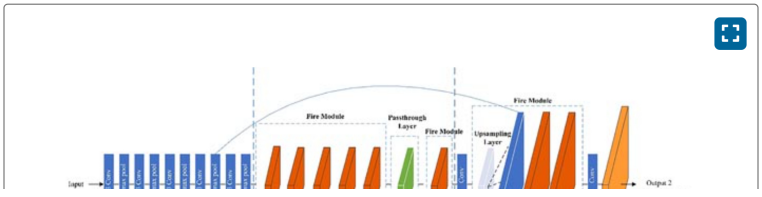
Citations

Keywords

Metrics

Abstract:

Deep neural networks (DNNs) have shown prominent performance in the field of object detection. However, DNNs usually run on powerful devices with high computational ability and sufficient memory, which have greatly limited their deployment for constrained environments such as embedded devices. YOLO is one of the state-of-the-art DNN-based object detection approaches with good performance both on speed and accuracy and Tiny-YOLO-V3 is its latest variant with a small model that can run on embedded devices. In this paper, Tinier-YOLO, which is originated from Tiny-YOLO-V3, is proposed to further shrink the model size while achieving improved detection accuracy and real-time performance. In Tinier-YOLO, the fire module in SqueezeNet is appointed by investigating the number of fire modules as well as their positions in the model in order to reduce the number of model parameters and then reduce the model size. For further improving the proposed Tinier-YOLO in terms of detection accuracy and real-time performance, the connectivity style between fire modules in Tinier-YOLO differs from SqueezeNet in that dense connection is introduced and fine designed to strengthen the feature propagation and ensure the maximum information flow in the network. The object detection performance is enhanced in Tinier-YOLO by using the passthrough layer that merges feature maps from the front layers to get fine-grained features, which can counter the negative effect of reducing the model size. The resulting Tinier-YOLO yields a model size of 8.9MB (almost 4× smaller than Tiny-YOLO-V3) while achieving 25 FPS real-time performance on Jetson TX1 and an mAP of 65.7% on PASCAL VOC and 34.0% on COCO. Tinier-YOLO alse posses comparable results in mAP and faster runtime speed with smaller model size and BFLOP/s value compared with other lightweight models like SqueezeNet SSD and MobileNet SSD.

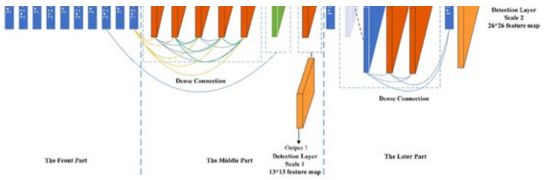


More Like This

Adaptive Convolution for Object Detection
IEEE Transactions on Multimedia
Published: 2019

Dual-Attention and Dynamic Convolution Enhanced YOLOv10 for Robust Outdoor Small Object Detection
2025 6th International Conference on Information Science and Education (ICISE-IE)
Published: 2025

Show More



The network structure of Tinier-YOLO.

Published in: IEEE Access (Volume: 8)

Page(s): 1935 - 1944 **DOI:** 10.1109/ACCESS.2019.2961959

Date of Publication: 24 December 2019 **Publisher:** IEEE

Electronic ISSN: 2169-3536

Funding Agency:

CCBY - IEEE is not the copyright holder of this material. Please follow the instructions via https://creativecommons.org/licenses/by/4.0/ to obtain full-text articles and stipulations in the API documentation.

SECTION I.
Introduction

Object detection is an important task in many popular fields such as medical diagnosis, robot navigation, automatic driving, augmented reality and so on. In these complex scenarios, object detection methods based on deep learning approach, such as Region-based Convolutional Neural Networks (R-CNN) [1], Spatial Pyramid Pooling Networks (SPPNet) [2], fast R-CNN [3], faster R-CNN [4], Region-based Fully Convolutional Networks (R-FCN) [5], Feature Pyramid Networks (FPN) [6], and You Only Look Once (YOLO) [7] show greater advantages than traditional methods. YOLO is one of the fastest object detection methods with good real-time performance and high accuracy, and it has been improved since it was proposed, including YOLO-V1, YOLO-V2, YOLO-V3. YOLO-V1 has two full-connected layers and twenty-four convolutional layers. The model size of YOLO-V1 has reached 1 GB, which occupies very large storage space and requires the running platform with high performance. YOLO-V2 [8] removes the fully-connected layers and introduces anchor boxes to predict bounding boxes, making the YOLO detector faster and more robust than YOLO-V1. YOLO-V3 [9] uses the residual structure to further deepen the network layer and achieves a breakthrough in accuracy. By running on the powerful GPU platform, YOLO and its improvements have reached high accuracy and fast speed. However, the model sizes of these object detection algorithms are too large for constrained environments with limited storage memory devices and they cannot work in constrained environments with real-time performance. Tiny-YOLO-V3 [9] is the latest improvement of YOLO with a relatively small model size for constrained environments. However, its detection accuracy is not high and the real-time performance is still not satisfactory on low computing power devices.

In order to get a more efficient object detection model for constrained environments, originated from Tiny- YOLO-V3, Tinier-YOLO is proposed in this paper to reduce the model size while achieving improved detection accuracy and real-time performance. Tinier-YOLO draws inspiration from the fire module in SqueezeNet to reduce the number of model parameters that helps to shrink the model size. One of the key challenges to introduce fire module in Tiny-YOLO-V3 is to investigate the number of fire modules as well as their

positions in the model. Another key challenge is to determine the connectivity style between fire modules in order to further get the improved detection accuracy and real-time performance. Motivated by the fine characteristic of dense connection in DenseNet [10], we adopt the dense connections between fire modules in Tinier-YOLO to strengthen the feature propagation and ensure the maximum information flow in the network. Since the reduction in model size, it will inevitably affect the detection accuracy. Therefore the passthrough layer is utilized in Tinier-YOLO to address this issue that can merge feature maps from the front layers to get fine-grained features. At last, in order to reduce the computational cost, we propose to remove batch normalization from the fire modules of Tinier-YOLO while trying to keep the overall performance.

The paper is organized as follows. Related work about object detection based on deep learning and network compression are introduced in Section 2. In Section 3, Tinier-YOLO is proposed including the motivation and the structure of Tinier-YOLO, where the front convolutional layers, fire modules, the dense connection between the fire modules, fine-grained features, and the batch normalization of Tinier-YOLO are introduced. In Section 4, experimental results and discussions are given, where the billion floating point operations per second (BFLOP/s), the model size, mean average precision (mAP), and the real-time performance of frames per second (FPS) are compared. Furthermore, the comparison results between Tinier-YOLO with other state-of-the-art lightweight models are given in Section 4. Finally, the conclusion is drawn in Section 5.

SECTION II.
Related Work

A. Object Detection Algorithms Based on Deep CNN

Early object detection methods such as Deformable Parts Models (DPM) [11] utilized handcrafted features of objects for detection. However, these features cannot represent object features in most complex scenes. AlexNet [12] based on CNN proposed by Krizhevsk in 2012 made a great breakthrough in the field of image classification. Alexnet provided a new idea for object detection which was considered as the extension work of image classification. Subsequently, CNN is widely used the field of object detection with its powerful ability to extract features automatically. Object detection algorithms based on deep CNN are generally classified into two categories, which are two-stage detectors and one-stage detectors.

Two-stage detectors divide the detection progress into generating candidate boxes and making predictions according to candidate boxes. R-CNN [1] is one of the leading two-stage object detectors. R-CNN applied high-capacity CNN to bottom-up region proposals in order to localize and segment objects. The features from candidate windows were extracted via deep convolutional networks. SPPNet [2] also showed great strength in object detection by generating a fixed-length representation regardless of image size/scale. Fast R-CNN [3] and Faster R-CNN [4] did not opted to use featurized image pyramids under default settings. Image pyramids were not the only way to compute a multi-scale feature representation. FPN [6] exploited the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids with marginal extra cost. R-FCN [5] proposed position-sensitive score maps to address a dilemma between translation-invariance in image classification and translation-variance in object detection and it reduced the computational redundancy of Faster R-CNN. The detection speed and accuracy of two-stage detectors are getting better and better, but it still cannot meet the real-time requirement of many actual scenes. The development of two-stage detectors gradually integrates the independent modules such as generating candidate boxes, feature extraction and bounding boxes regression in the object

detection system into a unified end-to-end learning framework.

Without generating candidate boxes in the first stage, one-stage detectors regard location information as a potential object, and then try to classify each area as the background or target object. YOLO [7] was proposed in 2016, which formulates object detection problem as a regression problem, making it a milestone of one-stage detectors. Compared with two-stage detectors, YOLO was extremely fast. However, it performed worse in location and detecting objects of small size. YOLO-V2 [8], YOLO-V3 [9] and the series of Single Shot Multi-box Detector (SSD) [13]– [16] have tried to solve the problem. SSD discretized the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. RetinaNet [17] introduced the focal loss function to deal with the problem of class imbalance and the accuracy is more higher. Recent work of anchor-free one-stage detectors like FSAF [18] and CenterNet [19] achieved a better trade-off between speed and accuracy. One-stage detectors have become more and more popular and been widely applied in the engineering field.

B. Network Compression

Building small and efficient network models for constraint environments has become a research hotspot in recent years. Related work on model compression and acceleration mainly includes network pruning and sparsity [20], network quantization [21]– [23] knowledge distillation [24], compact network architecture design and so on.

Network pruning and sparsity is to prune unimportant neuron connection. Quantization converts floating point calculation into bit calculation. Knowledge distillation technology utilizes the teacher model to help the student model to learn and improve the accuracy of the small model. The methods above mainly make modifications on the trained model. A better approach is to choose compact network architecture at the initial stage of model construction. Google Inception V1 [25] won the first place in ILSVRC 2014 games with a great advantage. The network structure of Google Inception V1 is largely based on Network in Network (NIN) [26]. The consolidation process of NIN cascading across features makes the network learn more complex and useful features. The 1×1 convolutional layer can not only organize information across channels but also improve the network expression ability and change the dimension of output channels. Inception V2 [27], Inception V3 [28], and Inception V4 [29] are the improvements of Inception V1, which make the model with less parameters, less computation, deeper network structure, and better performance. Xception [30] combined depth-wise convolution with Inception v3, which achieved higher performance with the same number of parameters, providing another way for designing lightweight network.

The SqueezeNet [31] proposed by UC Berkeley and Stanford researchers introduced fire module. The idea of fire module was similar to the series of Inception. It utilized 1*1 convolutional layer to compress the dimension of feature maps with the purposed to reduce parameters. The classification accuracy of SqueezeNet on ImageNet was close to AlexNet, but the model size was reduced by nearly 500 times.

ShuffleNet v1 [32] was proposed by Face++ for mobile devices. The main idea of ShuffleNet was to introduce pointwise group convolution and bottleneck-like module to evenly mix the feature maps after convolution according to channels. Then the number of channels for each convolution kernel and the computation complexity can be decreased. ShuffleNet v2 [33] utilized operations like channel split to improve information exchange between channels.

MobileNet v1 [34] divided traditional convolution into depthwise separable convolution and pointwise convolution. Computational time and the number of model parameters were greatly reduced while keeping high accuracy. The main contribution of MoblileNet v2 [35] was to add linear bottlenecks and inverted

residuals on the basis of MobileNet v1. MobileNet v3 [36] was built by combining Network Architecture Search (NAS) technology and NetAdapt algorithm. Before MobileNet v3 was proposed, Google Brain team proposed MnasNet [37] for mobile platforms. MnasNet incorporated speed information into the main reward function so that searches could identify a good trade-off between accuracy and speed. MobileNet v3 utilized the same search space based on RNN controller and decomposition as MnasNet, minimizing model latency while maintaining precision.

SECTION III.
The Proposed Tinier-YOLO for Constrained Environments

A. Motivation

Fig. 1 shows the network structure of Tiny-YOLO-V3, which is composed of seven convolutional layers and six maxpool layers for extracting image features and two scales of detection layers. Tiny-YOLO-V3 uses many convolutional layers with 512 and 1024 convolution filters, which results in a large number of parameters, large storage usage and slow detection speed on constrained environments. Another problem of Tiny-YOLO-V3 is that the detection accuracy is not high and the unreasonable compression methods in the network may further reduce the detection accuracy.

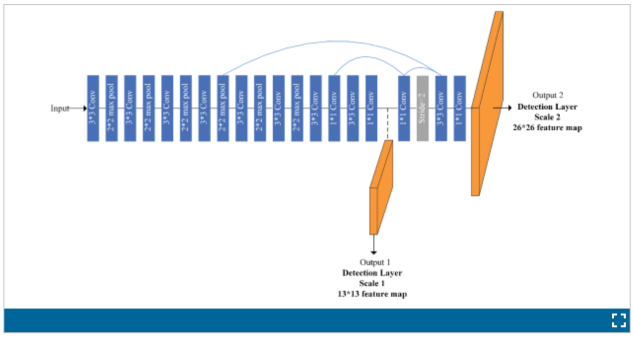


FIGURE 1.
The network structure of Tiny-YOLO-V3.

Show All

To address the problems in Tiny-YOLO-V3, Tinier-YOLO is proposed in this paper and focuses on the network performance of the model size, the detection speed and the accuracy. The purpose of Tinier-YOLO is to get a smaller, faster and better model that can run on constrained environments. First of all, the network structure is optimized by reducing the number of parameters reasonably instead of just deleting the convolutional layers blindly. The fire module of SqueezeNet uses the bottleneck layer network to compress the model and the network module is widen without losing detection accuracy heavily. Therefore, the fire module is introduced in Tinier-YOLO in order to get a smaller and faster network structure. And then, Tinier-YOLO seeks the way to get higher detection accuracy while reducing the number of parameters. Reference [10] indicates that the neural network is not necessarily progressive, which means a layer in the network not only relies on features from the adjacent layer but also depends on the features of the previous layers. For example, in a network with random depth, when the l th layer is removed, the $l+1$ th layer is connected directly to the $l-1$ th layer. When the second layer to the l th layer are removed, the $l+1$ th layer directly uses the features from the first layer, which is very similar to the dense connection in DenseNet and

the passthrough layer in YOLO-V2. The dense connection between fire modules and the passthrough layer are then properly used in Tinier-YOLO with the purpose to get higher detection accuracy.

B. Structure of Tinier-YOLO

Fig. 2 illustrates the structure of Tinier-YOLO. The front part of Tinier-YOLO retains the front five convolutional layers of Tiny-YOLO-V3. In the middle part of Tinier-YOLO, the five fire modules are introduced to implement network parameters compression and the dense connection between the fire modules is employed. It also merges previous feature maps with the passthrough layer before the first detection layer and predicts bounding boxes with the first scale. In the latter part of Tinier-YOLO, it adds two fire modules to process the combined feature maps and predicts bounding boxes with the second scale to get the fine-grained features. The architecture of Tinier-YOLO is darknet and the implementation details are as the following.

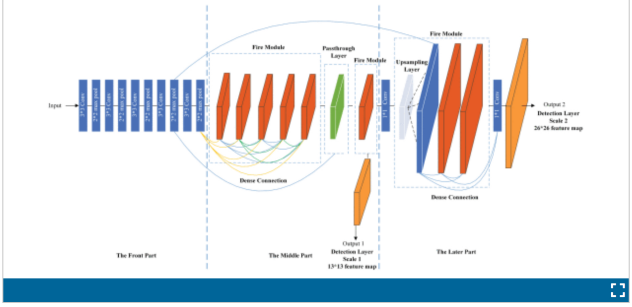


FIGURE 2.
The network structure of Tinier-YOLO. Tinier-YOLO retains the front five convolutional layers and detection layer of two different scales. The fire modules, the dense connections between the fire modules, and the passthrough layer are employed in Tinier-YOLO.

Show All

1) Fire Module of Tinier-YOLO

The fire module is introduced to reduce the number of model parameters and increase the depth and width of the whole network to ensure the detection accuracy. Fire module is comprised of the squeeze part and the expand part to compress and expand the data respectively. The squeeze part uses the 1×1 convolutional layer to replace the usual 3×3 convolutional layer. The 1×1 convolutional layer proposed by NIN is a very effective method to reduce the number of parameters. Since the 1×1 convolutional layer only has one parameter to be trained and learned, the detection accuracy does not decrease too much. In the expand part, both a convolutional layer with 1×1 filters and a layer with 3×3 filters are used. Then the outputs of these layers are mixed by the concatenation layer.

For a convolutional layer, the number of the input channels is c_i , the kernel size is k , and the number of the output channels is c_o . Then, the number of parameters of the convolutional layer is calculated in (1). For the fire module, the overall number of input channels is c_i . In the squeeze part, the kernel size of the squeeze part is k_{s_1} , and the number of the output channels is s_1 . When k_{s_1} is set to 1, a large number of parameters of the squeeze part can be reduced. For the expand part, the number of input channels is s_1 , the kernel sizes are k_{e_1} and k_{e_3} , respectively. The numbers of output channels are e_1 and e_3 , respectively. The number of output channel is $e_1 + e_3$. The number of parameters of the fire module is calculated in (2). The structure and specific parameters of the fire module in Tinier-YOLO are shown in Fig. 3.

$$\begin{aligned} P_{conv} &= (c_i \times k^2 + 1) \times c_o \\ P_{fire} &= (c_i \times k_{s_1}^2 + 1) \times s_1 + (s_1 \times k_{e_1}^2 + 1) \end{aligned} \tag{1}$$

$$\times e_1 + (s_1 \times k_{e_3}^2 + 1) \times e_3 \quad (2)$$

View Source

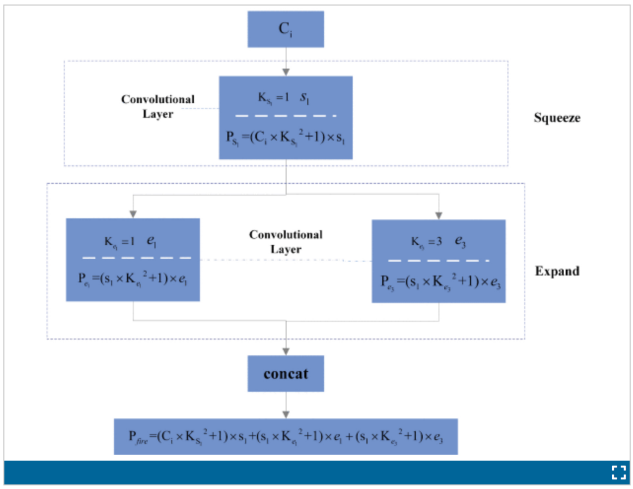


FIGURE 3.
The structure and specific parameters of the fire module.

Show All

In order to use the fire module more efficiently, the position of fire module is embedded in the network appropriately. There are eight fire modules in Tinier-YOLO. The 6th and 7th convolutional layers with 512 and 1024 filters in Tiny-YOLO-V3 are replaced with the first five fire modules. The convolutional layer with 512 filters before the first detection layer is replaced with a fire module. The convolutional layer with 256 filters before the second detection layer is replaced with the 7th and 8th fire modules. The number of input channels c_i is not limited, but more input channels bring more parameter reduction. As shown in Table 1, the reduced parameters of input channels of 512 are less than the convolutional layer with the input channels of 1024. To further reduce the number of parameters, it is appropriate to replace the convolutional layer with more input channels with fire module and these layers are distributed in the middle and latter parts of Tiny-YOLO-V3.

TABLE 1 Comparison of the Number of Parameters of Convolutional Layers and Fire Modules

	c_i	k	c_o	parameters(convolutional layer)	parameters(fire module)
Conv1	3	3	16	448	**184**
Conv2	16	3	32	4640	**740**
Conv3	32	3	64	18496	**2888**
Conv4	64	3	128	73856	**11408**
Conv5	128	3	256	295168	**45344**
Conv6	256	3	512	1180160	**180800**
Conv7	512	3	1024	4719616	**722048**
Conv8	1024	1	256	262400	**74016**
Conv9	256	3	512	1180160	**53536**
Conv10	384	3	256	884992	**53536**

We also found that if all the convolutional layers are replaced by the fire modules, the detection accuracy is too low since some convolutional layers with a small number of filters are replaced by the fire modules. If the first five convolutional layers with a small number of convolution filters less than 256 are retained rather than be replaced by the fire modules, the accuracy rate is increased by 6.2% and the model size is only increased by 1.6 MB. Therefore, Tinier-YOLO keeps the front convolutional layers and only replaces the five

convolutional layers in the middle part and the latter part of Tiny-YOLO-V3 with eight fire modules.

In general, a simple idea to compress the network is to reduce the number of network layers and the scale of the network and to use the shallow networks. However, the expressiveness capability of shallow netowork is far away from the DNN models [38]. Ba et. al. proposed to train shallow neural nets to mimic deeper models and got the similar performance with deep models, but the number of parameters has been increased [39]. In [40], Poole *et. al.* proved that the expressiveness capability can be grown exponentially with the depth increasing of network, but too shallow networks cannot replace deep networks effectively. As shown in Fig.2, after the five convolutional layers and five pooling layers, there are eight fire modules with the depth of two and several convolutional layers with 1*1 kernel in Tinier-YOLO. The last five convolutional layers in Tiny-YOLO-V3 have been removed and replaced by eight fire modules in Tinier-YOLO. The total network depth of Tinier-YOLO reaches 30, which is eleven layers deeper than that of Tiny-YOLO-V3. The accuracy of the network has therefore been increased. At the same time, BFLOP/s of these layers is decreased significantly from 6.93 to 0.65, where BFLOP/s is usually used to measure the time complexity by evaluating the number of model operations.

2) Dense Connection Between the Fire Modules

Dense connection is deployed between the fire modules to improve the accuracy by strengthening feature extraction ability and ensuring the maximum information flow in the network.

In SqueezeNet, fire modules feed-forward networks connect the output of the l th layer as input to the $l+1$ th layer, as in (3). The input of the l th module is represented as x^l_{fm} , the weight of the convolution kernel is denoted w^l_{fm} , b^l_{fm} is the bias parameter, * is the convolution operation, and $H^l_{fm}(.)$ is the activation function of the l th module. In Tinier-YOLO, $H^l_{fm}(.)$ is the common leaky ReLU activation. By introducing the idea of DenseNet [10], the l th fire module of Tiner-YOLO receives the feature maps of all preceding fire modules as in (4), $[x^0_{fm}, x^1_{fm}, \ldots, x^{l-1}_{fm}]$ refers to the concatenation of the feature maps produced in the fire modules.

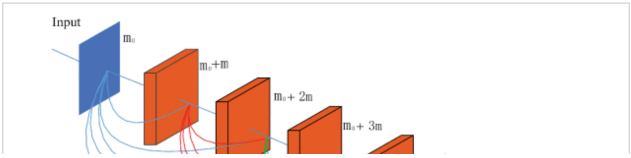
$$x^l_{fm} = H\left(x^{l-1}_{fm} * w^l_{fm} + b^l_{fm}\right) \tag{3}$$

$$x^l_{fm} = H\left(\left[x^0_{fm}, x^1_{fm} \ldots x^{l-1}_{fm}\right] * w^l_{fm} + b^l_{fm}\right) \tag{4}$$

View Source

Comparing (3) and (4), the l th fire module with dense connection always contains feature maps from the previous $l-1$ fire modules, which can strengthen the feature propagation in the network and is expected to improve accuracy.

Fig. 4 shows the dense connection between five fire modules in Tinier-YOLO. The feature maps of the first $l-1$ fire modules are concatenated and utilized as the input of the l th fire module. m_0 is the feature map of the previous convolutional layer and m is the feature map of the fire modules. Therefore the l th fire module outputs $m_0 + m(l-1)$ feature maps. The feature maps are relatively smaller that with fewer parameters and calculations.



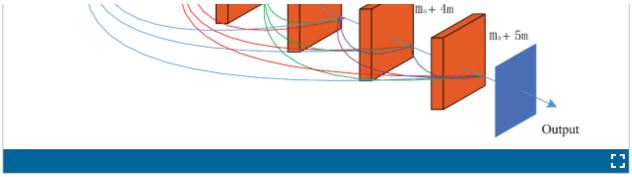


FIGURE 4.
Dense connection between the fire modules.

Show All

If the dense connection is deployed in larger feature maps or connects the feature maps with different size, it will result in a huge amount of calculation, which greatly affects the real-time performance. We have tested the real-time performance of four networks with different dense connections. As shown in Table 2, the proposed dense connection in this paper has the best real-time performance with 25.1 FPS.

TABLE 2 Comparison of the Real-Time Performance

Connection Type	Real-time performance(FPS)
Tiny-YOLO-V2	15.5
Tiny-YOLO-V3	**21.6**
Between convolutional layers	13
Between fire modules	**25.1**

3) Retaining Fine-Grained Features

In Tinier-YOLO, the passthrough layer is utilized to achieve the combination of multi-scale features. Passthrough layer is added in Tinier-YOLO to acquire the upper level features of 26×26 and combine this feature with the final 13×13 output features. Tinier-YOLO combines the passthrough layer with the output of the 5th fire module and the 5th convolutional layer, which does not bring too many parameters and computations and improves the detection accuracy. The passthrough layer can fuse the underlying features of the network and also improve the ability of the model to detect small targets. At the same time, the boxes are predicted at two different scales which are 13×13 and 26×26 and the passthrough layer is utilized to take the feature maps from the previous fire module and upsample it by $2\times$ just like that in Tiny-YOLO-V3. Then two fire modules are added to process this combined feature maps and eventually a similar tensor is predicted.

4) Batch Normalization in Tinier-YOLO

Since the distribution of each layer's inputs in DNNs changes during training, batch normalization is proposed to address this problem and has been widely used in training DNNs that helps to use much higher learning rates [27]. Before inputting to the next layer, batch normalization takes the output of one layer and transforms its mean variance and standard deviation.

Batch normalization was introduced in YOLO series since the version of YOLO-V2. However, the computational cost has been increased 30% if batch normalization is used in every layers since means and variances of batch normalization are adjusted constantly during training [27]. The same problem also arises in Tiny-YOLO-V3. In the proposed Tinier-YOLO, fire modules have been involved with batch normalization being operated in each convolutional layer. In order to reduce the computational cost, we tried to remove batch normalization from the fire modules of Tinier-YOLO. We found that the training time for the model has been decreased after removing batch normalization in fire modules. And we also carried out the test on a total of 4952 images to evaluate the performance of Tinier-YOLO with batch normalization in every layers (Tinier-YOLO-BN) and with batch normalization

in every layers except that in the fire modules (Tinier-YOLO). Table 3 gives the results of average test time, mAP, and model size. The results show that the detection accuracy has not been changed while the real-time performance and model size have slight improvement after the batch normalization is removed from the layers in the fire modules of Tinier-YOLO.

TABLE 3 Comparison Between Tinier-YOLO With and Without Batch Normalization

	Average test time (s)	mAP	Model size (MB)
Tinier-YOLO	**195**	65.7%	**8.8**
Tinier-YOLO-BN	197	65.7%	8.9

SECTION IV.
Experimental Results

In this section, the experimental settings and experimental datasets are introduced firstly. Then the performance comparison under the metrics of mAP, BFLOP/s, model size, and FPS are carried out. Furthermore, in order to verify the improvement of Tiner-YOLO compared with the other lightweight architectures, we conduct extensive experiments to illustrate the performance comparison results.

A. Experimental Settings

Tinier-YOLO is trained on NVIDIA Titan X in order to save the training time. The detection speed is tested on Jetson TX1, which is an embedded AI device built around the revolutionary NVIDIA Maxwell architecture with 256 CUDA cores delivering over 1 TeraFLOPs of performance. Jetson TX1 is suitable for embedded deep learning, computer vision, graphics, and GPU computing. Table 4 gives the configuration comparison between Jetson TX1 and TITAN X. There is only 16GB data storage of Jetson TX1, which is even smaller than normal mobile phones. The default power mode of Jetson TX1 is used for testing the detection speed.

TABLE 4 Comparison of Configuration Between Titan X and Jetson TX1

Computing platform	GPU	Data Storage	Memory
Titan X	NVIDIA Pascal, 3584 CUDA cores	1TB	10GB 384bit GDDR5X
Jetson TX1	NVIDIA Maxwell, 256 CUDA cores	16 GB	4 GB 64 bit LPDDR4

The source codes and architecture of Tinier-YOLO is available for download here: https://github.com/jnfrancis/Tinier-YOLO.

B. Experimental Datasets

The datasets of PASCAL VOC [41] and COCO [42] are used in this paper. These two datasets are usually used for image classification, object detection, and image segmentation. The dataset of PASCAL VOC includes PASCAL VOC 2007 and PASCAL VOC 2012. There are 20 kinds of objects with 29503 images. When calculating the AP values and mAP, the default value of 0.5 is used for the intersection over union (IOU). The dataset of COCO is more difficult to train than PASCAL VOC. For object detection algorithms, the performance of a model is usually more inclined on COCO. There are 80 categories of objects in COCO and the number of targets contained in each image is three times more than PASCAL VOC. The mAP value of each kind of object is the average by 10 values of IOU from 0.5 to 0.95 and mAP value is also compared on small, medium, and large size of object. The performance evaluation by multiple IOU values on

COCO can better reflect the comprehensive performance of the algorithm. Table 5 shows the comparison between PASCAL VOC and COCO in detail.

TABLE 5 Detailed Comparison Between PASCAL VOC and COCO

Dataset	PASCAL VOC	COCO
Number of classes	20	80
Training dataset	16551	117264
Test dataset	4952	5000
Number of ground truth boxes	52090	902435
Number of boxes per image	2.4	7.4

C. Model Training

The darknet53.conv.74 is loaded in Tinier-YOLO before training on both datasets which is the same as that in Tiny-YOLO-V3. The darknet53.conv.74 is the pre-trained weight model based on the darknet53 [9]. The input size of Tinier-YOLO is 416*416.

The number of maximum training batches of Pascal VOC is 500K and that of COCO is 2000K, which are set according to the number of classes of each dataset. The batch size of Tinier-YOLO is 64, the subdivision is 8, the momentum is 0.9 and the decay is 0.005. In order to make the model converge much faster in the early training period, the learning rate is 0.001 and then decreased to 0.1 times of the original as the number of iteration batch reaches 400K and 450K.

D. Ablation Experiments for Tinier-YOLO

In order to show the effects of the proposed techniques on the performance of Tinier-YOLO, we carried out the ablation experiments by adding fire module, dense connection, and passthrough layer successively in the network of Tinier-YOLO. Table 6 gives the results of mAP, BFLOP/s, FPS, and model size by the ablation experiments.

TABLE 6 Ablation Experiments From Tiny-YOLO-V3 to Tinier-YOLO on PASCAL VOC

	Tiny-YOLO-V3			Tinier-YOLO
Fire Module		✓	✓	✓
Dense Connection			✓	✓
Passthrough Layer				✓
Model Size(MB)	34.9	**6.9**	8.3	8.9
BFLOP/s	5.474	**2.350**	2.552	2.563
mAP(%)	61.3	60.6	**64.8**	**65.7**
FPS	21.6	25.5	24.9	**25.1**

As shown in Table 6, compared with Tiny-YOLO-V3, the introducing of fire module reduces the model size and BFLOP/s greatly without decreasing the detection accuracy significantly. The model size is decreased from 34.9 MB of Tiny-YOLO-V3 to 6.9 MB while keeping high detection accuracy. The employment of dense connection between the fire modules and passthrough layer makes a great contribution to the improvement in mAP without increasing too many parameters. Employing dense connection increases the mAP by 3.5% on the basis of introducing fire modules. Utilizing the passthrough layer increases mAP by about 1% on the basis of introducing dense connection between the fire modules.

1) Performance Comparison on BFLOP/s and the Model Size

BFLOP/s is often used as a measure of the computational power required by a model. The larger the BFLOP/s, the higher the requirements for the device. As shown in Table 7, BFLOP/s of Tinier-YOLO is only 2.563, which is less than the

half of Tiny-YOLO-V3 and is suitable for constrained environment. The model size of Tinier-YOLO is almost 4× smaller than Tiny-YOLO-V3, 7× smaller than Tiny-YOLO-V2.

TABLE 7 Performance of BFLOP/s and Model Size

	YOLO-V1	YOLO-V2	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
Size	1.1G	202.7MB	63.4MB	34.9MB	**8.9MB**
BFLOP/s	40.19	34.9	6.97	5.474	**2.563**

2) Comparison on Real-Time Performance

The reference point of measuring real-time performance is FPS and depends on the resolution of the test video or the image. With the support of Jetson TX1, we measured the total time for testing 4952 images in the VOC 2007 dataset. The average value of testing time by 30 times is shown in Table 8. It can be seen that the total time required for the Tinier-YOLO to detect about 5,000 pictures is less than 200 seconds. Tinier-YOLO performs faster than Tiny-YOLO-V3. Tiny-YOLO-V2 takes more than 320 seconds to complete the detection. Tinier-YOLO can detect objects with 25.1 FPS. The real-time performance is increased by 16.2% compared with Tiny-YOLO-V3 and increased by 39.4% compared with Tiny-YOLO-V2.

TABLE 8 Comparison of the Real-Time Performance

	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
Average test time(s)	320.2	229.0	**196.9**
FPS	15.5	21.6	**25.1**

3) Performance Comparison on mAP

mAP is usually used to measure the detection accuracy. On the VOC 2007 dataset, the AP values of the three compared methods are shown in Table 9. From the average AP values, it can be seen that Tinier-YOLO improved the detection accuracy by 4.5% compared with Tiny-YOLO-V3, and nearly 8.6% compared with Tiny-YOLO-V2. The AP values of all the 20 classes are better than Tiny- YOLO-V2. Compared with Tiny-YOLO-V3, except for the class of aeroplane, the AP values of the other 19 classes are all better than Tiny-YOLO-V3, which demonstrates the effectiveness of Tiner-YOLO.

TABLE 9 The AP Values of Three Methods on PASCAL VOC Dataset

	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
aeroplane	64.0	**69.9**	67.8
bicycle	74.7	75.2	**77.6**
bird	48.7	44.6	**51.5**
boat	41.8	50.4	**53.8**
bottle	18.2	33.0	**43.8**
bus	70.1	73.0	**76.8**
car	69.4	77.5	**80.9**
cat	72.7	68.4	**77.8**
chair	33.4	38.6	**44.7**
cow	53.9	60.0	**68.8**
diningtable	58.5	59.2	**61.2**
dog	63.6	61.2	**67.9**
horse	73.8	75.6	**80.3**
motorbike	71.4	75.8	**76.3**
person	61.6	71.6	**75.4**
pottedplant	25.6	28.4	**38.8**
sheep	56.1	64.0	**66.0**
sofa	52.8	58.8	**62.2**
train	72.9	75.1	**75.9**
tvmonitor	60.1	65.0	**66.8**
avg	57.1	61.3	**65.7**

On the COCO dataset, the mAP results of Tiny-YOLO-V2, Tiny-YOLO-V3, and Tinier-YOLO are shown in Table 10. The mAP value under IOU = 0.5 of Tinier-

YOLO is almost 1% higher than that of Tiny-YOLO-V3. In case of more strict tests under IOU = 0.75 and IOU from 0.50 to 0.95, the mAP values of Tinier-YOLO are 1.7% and 3.3% higher than Tiny-YOLO-V3. The Average Recall (AR) of Tinier-YOLO is also better than Tiny-YOLO-V3. Table 10 also indicates that Tinier-YOLO makes more improvements in detecting medium and large objects than detecting small objects.

TABLE 10 Performance of mAP on the COCO Dataset

	IOU	area	maxDets	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
mAP(%)	0.50:0.95	all	100	9.9	15.3	**17.0**
mAP(%)	0.50	all	100	23.7	33.1	**34.0**
mAP(%)	0.75	all	100	6.8	12.4	**15.7**
mAP(%)	0.50:0.95	small	100	7.0	4.4	**4.8**
mAP(%)	0.50:0.95	medium	100	7.8	15.2	**17.3**
mAP(%)	0.50:0.95	large	100	20.1	25.1	**26.8**
AR(%)	0.50:0.95	all	1	12.4	16.7	**18.5**
AR(%)	0.50:0.95	all	10	19.4	25.9	**28.3**
AR(%)	0.50:0.95	all	100	21.1	27.5	**30.0**
AR(%)	0.50:0.95	small	100	2.3	7.7	**8.4**
AR(%)	0.50:0.95	medium	100	20.4	30.3	**33.9**
AR(%)	0.50:0.95	large	100	41.1	44.4	**47.2**

4) Performance Comparison Between Tinier-YOLO and Other Lightweight Models

In order to further evaluate the improvement of Tiner-YOLO comparing to the other state-of-the-art lightweight architectures, we take SqueezeNet SSD and MobileNet SSD [43] into comparison. The models of SqueezeNet SSD and

PDF
Help

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 30, 2019, accepted December 16, 2019, date of publication December 24, 2019,
date of current version January 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2961959

Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments

WEI FANG[1,2], (Member, IEEE), LIN WANG[1], AND PEIMING REN[1]
[1]School of IoT Engineering, Jiangnan University, Wuxi 214122, China
[2]Jiangsu Provincial Engineering Laboratory of Pattern Recognition and Computational Intelligence, Wuxi 214122, China

Corresponding author: Wei Fang (fangwei@jiangnan.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFC1601000 and Grant 2017YFC1601800, in part by the National Natural Science Foundation of China, under Grant 61673194 and Grant 61672263, in part by the Key Research and Development Program of Jiangsu Province, China, under Grant BE2017630, in part by the Blue Project in Jiangsu Universities, and in part by the Postdoctoral Science Foundation of China under Grant 2014M560390.

ABSTRACT Deep neural networks (DNNs) have shown prominent performance in the field of object detection. However, DNNs usually run on powerful devices with high computational ability and sufficient memory, which have greatly limited their deployment for constrained environments such as embedded devices. YOLO is one of the state-of-the-art DNN-based object detection approaches with good performance both on speed and accuracy and Tiny-YOLO-V3 is its latest variant with a small model that can run on embedded devices. In this paper, Tinier-YOLO, which is originated from Tiny-YOLO-V3, is proposed to further shrink the model size while achieving improved detection accuracy and real-time performance. In Tinier-YOLO, the fire module in SqueezeNet is appointed by investigating the number of fire modules as well as their positions in the model in order to reduce the number of model parameters and then reduce the model size. For further improving the proposed Tinier-YOLO in terms of detection accuracy and real-time performance, the connectivity style between fire modules in Tinier-YOLO differs from SqueezeNet in that dense connection is introduced and fine designed to strengthen the feature propagation and ensure the maximum information flow in the network. The object detection performance is enhanced in Tinier-YOLO by using the passthrough layer that merges feature maps from the front layers to get fine-grained features, which can counter the negative effect of reducing the model size. The resulting Tinier-YOLO yields a model size of 8.9MB (almost 4× smaller than Tiny-YOLO-V3) while achieving 25 FPS real-time performance on Jetson TX1 and an mAP of 65.7% on PASCAL VOC and 34.0% on COCO. Tinier-YOLO alse posses comparable results in mAP and faster runtime speed with smaller model size and BFLOP/s value compared with other lightweight models like SqueezeNet SSD and MobileNet SSD.

INDEX TERMS Constrained environments, dense connection, fire modules, passthrough layer, YOLO.

I. INTRODUCTION

Object detection is an important task in many popular fields such as medical diagnosis, robot navigation, automatic driving, augmented reality and so on. In these complex scenarios, object detection methods based on deep learning approach, such as Region-based Convolutional Neural Networks (R-CNN) [1], Spatial Pyramid Pooling Networks (SPPNet) [2], fast R-CNN [3], faster R-CNN [4], Region-based Fully Convolutional Networks (R-FCN) [5], Feature Pyramid Networks (FPN) [6], and You Only Look Once (YOLO) [7] show greater advantages than traditional methods. YOLO is one of the fastest object detection methods with good real-time performance and high accuracy, and it has been improved since it was proposed, including YOLO-V1, YOLO-V2, YOLO-V3. YOLO-V1 has two full-connected layers and twenty-four convolutional layers. The model size of YOLO-V1 has reached 1 GB, which occupies very large storage space and requires the running platform with high performance. YOLO-V2 [8] removes the fully-connected layers and introduces anchor boxes to predict bounding boxes, making the YOLO detector faster and more robust than YOLO-V1. YOLO-V3 [9] uses the residual structure to further deepen the network layer and achieves a breakthrough in accuracy. By running on the powerful GPU platform, YOLO and its improvements have reached high

The associate editor coordinating the review of this manuscript and approving it for publication was Joewono Widjaja.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

AIRSPACE SYS. INC. EX2073, Page 14 (IPR2025-00629)

accuracy and fast speed. However, the model sizes of these object detection algorithms are too large for constrained environments with limited storage memory devices and they cannot work in constrained environments with real-time performance. Tiny-YOLO-V3 [9] is the latest improvement of YOLO with a relatively small model size for constrained environments. However, its detection accuracy is not high and the real-time performance is still not satisfactory on low computing power devices.

In order to get a more efficient object detection model for constrained environments, originated from Tiny-YOLO-V3, Tinier-YOLO is proposed in this paper to reduce the model size while achieving improved detection accuracy and real-time performance. Tinier-YOLO draws inspiration from the fire module in SqueezeNet to reduce the number of model parameters that helps to shrink the model size. One of the key challenges to introduce fire module in Tiny-YOLO-V3 is to investigate the number of fire modules as well as their positions in the model. Another key challenge is to determine the connectivity style between fire modules in order to further get the improved detection accuracy and real-time performance. Motivated by the fine characteristic of dense connection in DenseNet [10], we adopt the dense connections between fire modules in Tinier-YOLO to strengthen the feature propagation and ensure the maximum information flow in the network. Since the reduction in model size, it will inevitably affect the detection accuracy. Therefore the passthrough layer is utilized in Tinier-YOLO to address this issue that can merge feature maps from the front layers to get fine-grained features. At last, in order to reduce the computational cost, we propose to remove batch normalization from the fire modules of Tinier-YOLO while trying to keep the overall performance.

The paper is organized as follows. Related work about object detection based on deep learning and network compression are introduced in Section 2. In Section 3, Tinier-YOLO is proposed including the motivation and the structure of Tinier-YOLO, where the front convolutional layers, fire modules, the dense connection between the fire modules, fine-grained features, and the batch normalization of Tinier-YOLO are introduced. In Section 4, experimental results and discussions are given, where the billion floating point operations per second (BFLOP/s), the model size, mean average precision (mAP), and the real-time performance of frames per second (FPS) are compared. Furthermore, the comparison results between Tinier-YOLO with other state-of-the-art lightweight models are given in Section 4. Finally, the conclusion is drawn in Section 5.

II. RELATED WORK

A. OBJECT DETECTION ALGORITHMS BASED ON DEEP CNN

Early object detection methods such as Deformable Parts Models (DPM) [11] utilized handcrafted features of objects for detection. However, these features cannot represent object features in most complex scenes. AlexNet [12] based on CNN proposed by Krizhevsk in 2012 made a great breakthrough in the field of image classification. Alexnet provided a new idea for object detection which was considered as the extension work of image classification. Subsequently, CNN is widely used the field of object detection with its powerful ability to extract features automatically. Object detection algorithms based on deep CNN are generally classified into two categories, which are two-stage detectors and one-stage detectors.

Two-stage detectors divide the detection progress into generating candidate boxes and making predictions according to candidate boxes. R-CNN [1] is one of the leading two-stage object detectors. R-CNN applied high-capacity CNN to bottom-up region proposals in order to localize and segment objects. The features from candidate windows were extracted via deep convolutional networks. SPPNet [2] also showed great strength in object detection by generating a fixed-length representation regardless of image size/scale. Fast R-CNN [3] and Faster R-CNN [4] did not opted to use featurized image pyramids under default settings. Image pyramids were not the only way to compute a multi-scale feature representation. FPN [6] exploited the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids with marginal extra cost. R-FCN [5] proposed position-sensitive score maps to address a dilemma between translation-invariance in image classification and translation-variance in object detection and it reduced the computational redundancy of Faster R-CNN. The detection speed and accuracy of two-stage detectors are getting better and better, but it still cannot meet the real-time requirement of many actual scenes. The development of two-stage detectors gradually integrates the independent modules such as generating candidate boxes, feature extraction and bounding boxes regression in the object detection system into a unified end-to-end learning framework.

Without generating candidate boxes in the first stage, one-stage detectors regard location information as a potential object, and then try to classify each area as the background or target object. YOLO [7] was proposed in 2016, which formulates object detection problem as a regression problem, making it a milestone of one-stage detectors. Compared with two-stage detectors, YOLO was extremely fast. However, it performed worse in location and detecting objects of small size. YOLO-V2 [8], YOLO-V3 [9] and the series of Single Shot Multi-box Detector (SSD) [13]–[16] have tried to solve the problem. SSD discretized the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. RetinaNet [17] introduced the focal loss function to deal with the problem of class imbalance and the accuracy is more higher. Recent work of anchor-free one-stage detectors like FSAF [18] and CenterNet [19] achieved a better trade-off between speed and accuracy. One-stage detectors have become more and more popular and been widely applied in the engineering field.

AIRSPACE SYS. INC. EX2073, Page 15 (IPR2025-00629)

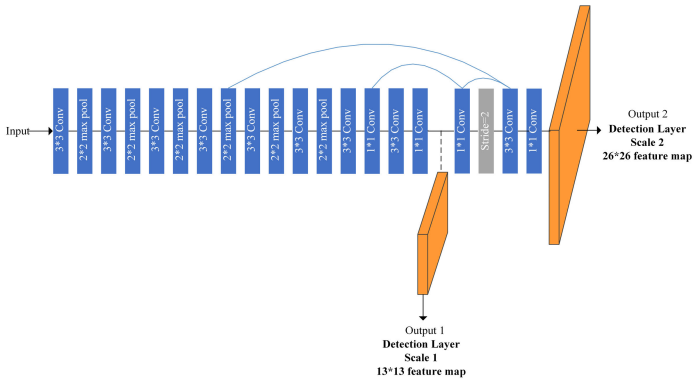


FIGURE 1. The network structure of Tiny-YOLO-V3.

B. NETWORK COMPRESSION

Building small and efficient network models for constraint environments has become a research hotspot in recent years. Related work on model compression and acceleration mainly includes network pruning and sparsity [20], network quantization [21]–[23] knowledge distillation [24], compact network architecture design and so on.

Network pruning and sparsity is to prune unimportant neuron connection. Quantization converts floating point calculation into bit calculation. Knowledge distillation technology utilizes the teacher model to help the student model to learn and improve the accuracy of the small model. The methods above mainly make modifications on the trained model. A better approach is to choose compact network architecture at the initial stage of model construction. Google Inception V1 [25] won the first place in ILSVRC 2014 games with a great advantage. The network structure of Google Inception V1 is largely based on Network in Network (NIN) [26]. The consolidation process of NIN cascading across features makes the network learn more complex and useful features. The 1×1 convolutional layer can not only organize information across channels but also improve the network expression ability and change the dimension of output channels. Inception V2 [27], Inception V3 [28], and Inception V4 [29] are the improvements of Inception V1, which make the model with less parameters, less computation, deeper network structure, and better performance. Xception [30] combined depth-wise convolution with Inception v3, which achieved higher performance with the same number of parameters, providing another way for designing lightweight network.

The SqueezeNet [31] proposed by UC Berkeley and Stanford researchers introduced fire module. The idea of fire module was similar to the series of Inception. It utilized 1*1 convolutional layer to compress the dimension of feature maps with the purposed to reduce parameters. The classification accuracy of SqueezeNet on ImageNet was close to AlexNet, but the model size was reduced by nearly 500 times.

ShuffleNet v1 [32] was proposed by Face++ for mobile devices. The main idea of ShuffleNet was to introduce pointwise group convolution and bottleneck-like module to evenly mix the feature maps after convolution according to channels. Then the number of channels for each convolution kernel and the computation complexity can be decreased. ShuffleNet v2 [33] utilized operations like channel split to improve information exchange between channels.

MobileNet v1 [34] divided traditional convolution into depthwise separable convolution and pointwise convolution. Computational time and the number of model parameters were greatly reduced while keeping high accuracy. The main contribution of MoblileNet v2 [35] was to add linear bottlenecks and inverted residuals on the basis of MobileNet v1. MobileNet v3 [36] was built by combining Network Architecture Search (NAS) technology and NetAdapt algorithm. Before MobileNet v3 was proposed, Google Brain team proposed MnasNet [37] for mobile platforms. MnasNet incorporated speed information into the main reward function so that searches could identify a good trade-off between accuracy and speed. MobileNet v3 utilized the same search space based on RNN controller and decomposition as MnasNet, minimizing model latency while maintaining precision.

III. THE PROPOSED Tinier-YOLO FOR CONSTRAINED ENVIRONMENTS

A. MOTIVATION

Fig. 1 shows the network structure of Tiny-YOLO-V3, which is composed of seven convolutional layers and six maxpool layers for extracting image features and two scales of detection layers. Tiny-YOLO-V3 uses many convolutional layers with 512 and 1024 convolution filters, which results in a large number of parameters, large storage usage and slow detection speed on constrained environments. Another problem of Tiny-YOLO-V3 is that the detection accuracy is not high and the unreasonable compression methods in the network may further reduce the detection accuracy.

To address the problems in Tiny-YOLO-V3, Tinier-YOLO is proposed in this paper and focuses on the network

AIRSPACE SYS. INC. EX2073, Page 16 (IPR2025-00629)

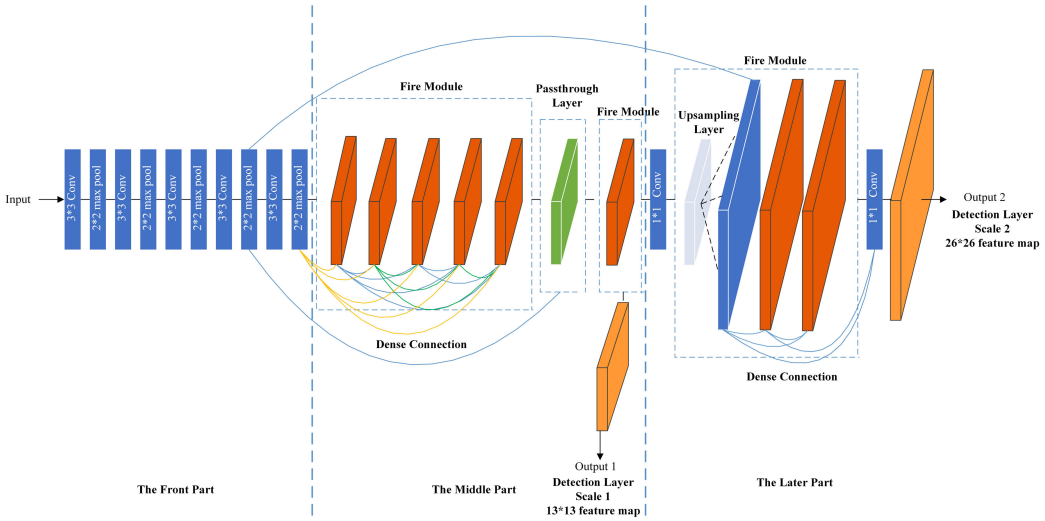


FIGURE 2. The network structure of Tinier-YOLO. Tinier-YOLO retains the front five convolutional layers and detection layer of two different scales. The fire modules, the dense connections between the fire modules, and the passthrough layer are employed in Tinier-YOLO.

performance of the model size, the detection speed and the accuracy. The purpose of Tinier-YOLO is to get a smaller, faster and better model that can run on constrained environments. First of all, the network structure is optimized by reducing the number of parameters reasonably instead of just deleting the convolutional layers blindly. The fire module of SqueezeNet uses the bottleneck layer network to compress the model and the network module is widen without losing detection accuracy heavily. Therefore, the fire module is introduced in Tinier-YOLO in order to get a smaller and faster network structure. And then, Tinier-YOLO seeks the way to get higher detection accuracy while reducing the number of parameters. Reference [10] indicates that the neural network is not necessarily progressive, which means a layer in the network not only relies on features from the adjacent layer but also depends on the features of the previous layers. For example, in a network with random depth, when the lth layer is removed, the $l+1$th layer is connected directly to the $l-1$th layer. When the second layer to the lth layer are removed, the $l+1$th layer directly uses the features from the first layer, which is very similar to the dense connection in DenseNet and the passthrough layer in YOLO-V2. The dense connection between fire modules and the passthrough layer are then properly used in Tinier-YOLO with the purpose to get higher detection accuracy.

B. STRUCTURE OF Tinier-YOLO

Fig. 2 illustrates the structure of Tinier-YOLO. The front part of Tinier-YOLO retains the front five convolutional layers of Tiny-YOLO-V3. In the middle part of Tinier-YOLO, the five fire modules are introduced to implement network parameters compression and the dense connection between the fire modules is employed. It also merges previous feature maps with the passthrough layer before the first detection layer and predicts bounding boxes with the first scale. In the latter part of Tinier-YOLO, it adds two fire modules to process the combined feature maps and predicts bounding boxes with the second scale to get the fine-grained features. The architecture of Tinier-YOLO is darknet and the implementation details are as the following.

1) FIRE MODULE OF Tinier-YOLO

The fire module is introduced to reduce the number of model parameters and increase the depth and width of the whole network to ensure the detection accuracy. Fire module is comprised of the squeeze part and the expand part to compress and expand the data respectively. The squeeze part uses the 1×1 convolutional layer to replace the usual 3×3 convolutional layer. The 1×1 convolutional layer proposed by NIN is a very effective method to reduce the number of parameters. Since the 1×1 convolutional layer only has one parameter to be trained and learned, the detection accuracy does not decrease too much. In the expand part, both a convolutional layer with 1×1 filters and a layer with 3×3 filters are used. Then the outputs of these layers are mixed by the concatenation layer.

For a convolutional layer, the number of the input channels is c_i, the kernel size is k, and the number of the output channels is c_o. Then, the number of parameters of the convolutional layer is calculated in (1). For the fire module, the overall number of input channels is c_i. In the squeeze part, the kernel size of the squeeze part is k_{s_1}, and the number of the output channels is s_1. When k_{s_1} is set to 1, a large number of parameters of the squeeze part can be reduced. For the expand part, the number of input channels is s_1, the kernel sizes are k_{e_1} and k_{e_3}, respectively. The numbers of output channels

AIRSPACE SYS. INC. EX2073, Page 17 (IPR2025-00629)

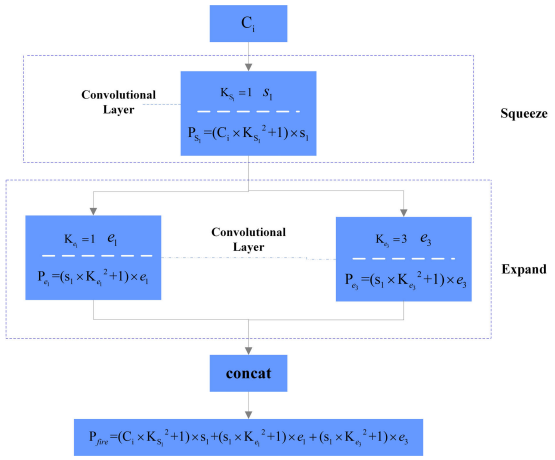


FIGURE 3. The structure and specific parameters of the fire module.

are e_1 and e_3, respectively. The number of output channel is $e_1 + e_3$. The number of parameters of the fire module is calculated in (2). The structure and specific parameters of the fire module in Tinier-YOLO are shown in Fig. 3.

$$P_{conv} = (c_i \times k^2 + 1) \times c_o \tag{1}$$

$$P_{fire} = (c_i \times k_{s_1}^2 + 1) \times s_1 + (s_1 \times k_{e_1}^2 + 1) \times e_1 + (s_1 \times k_{e_3}^2 + 1) \times e_3 \tag{2}$$

In order to use the fire module more efficiently, the position of fire module is embedded in the network appropriately. There are eight fire modules in Tinier-YOLO. The 6th and 7th convolutional layers with 512 and 1024 filters in Tiny-YOLO-V3 are replaced with the first five fire modules. The convolutional layer with 512 filters before the first detection layer is replaced with a fire module. The convolutional layer with 256 filters before the second detection layer is replaced with the 7th and 8th fire modules. The number of input channels c_i is not limited, but more input channels bring more parameter reduction. As shown in Table 1, the reduced parameters of input channels of 512 are less than the convolutional layer with the input channels of 1024. To further reduce the number of parameters, it is appropriate to replace the convolutional layer with more input channels with fire module and these layers are distributed in the middle and latter parts of Tiny-YOLO-V3.

We also found that if all the convolutional layers are replaced by the fire modules, the detection accuracy is too low since some convolutional layers with a small number of filters are replaced by the fire modules. If the first five convolutional layers with a small number of convolution filters less than 256 are retained rather than be replaced by the fire modules, the accuracy rate is increased by 6.2% and the model size is only increased by 1.6 MB. Therefore, Tinier-YOLO keeps the front convolutional layers and only replaces the five convolutional layers in the middle part and the latter part of Tiny-YOLO-V3 with eight fire modules.

TABLE 1. Comparison of the number of parameters of convolutional layers and fire modules.

	c_i	k	c_o	parameters(convolutional layer)	parameters(fire module)
Conv1	3	3	16	448	**184**
Conv2	16	3	32	4640	**740**
Conv3	32	3	64	18496	**2888**
Conv4	64	3	128	73856	**11408**
Conv5	128	3	256	295168	**45344**
Conv6	256	3	512	1180160	**180800**
Conv7	512	3	1024	4719616	**722048**
Conv8	1024	1	256	262400	**74016**
Conv9	256	3	512	1180160	**53536**
Conv10	384	3	256	884992	**53536**

In general, a simple idea to compress the network is to reduce the number of network layers and the scale of the network and to use the shallow networks. However, the expressiveness capability of shallow netowork is far away from the DNN models [38]. Ba et. al. proposed to train shallow neural nets to mimic deeper models and got the similar performance with deep models, but the number of parameters has been increased [39]. In [40], Poole *et. al.* proved that the expressiveness capability can be grown exponentially with the depth increasing of network, but too shallow networks cannot replace deep networks effectively. As shown in Fig.2, after the five convolutional layers and five pooling layers, there are eight fire modules with the depth of two and several convolutional layers with 1*1 kernel in Tinier-YOLO. The last five convolutional layers in Tiny-YOLO-V3 have been removed and replaced by eight fire modules in Tinier-YOLO. The total network depth of Tinier-YOLO reaches 30, which is eleven layers deeper than that of Tiny-YOLO-V3. The accuracy of the network has therefore been increased. At the same time, BFLOP/s of these layers is decreased significantly from 6.93 to 0.65, where BFLOP/s is usually used to measure the time complexity by evaluating the number of model operations.

2) DENSE CONNECTION BETWEEN THE FIRE MODULES

Dense connection is deployed between the fire modules to improve the accuracy by strengthening feature extraction ability and ensuring the maximum information flow in the network.

In SqueezeNet, fire modules feed-forward networks connect the output of the lth layer as input to the $l + 1$th layer, as in (3). The input of the lth module is represented as x_{fm}^l, the weight of the convolution kernel is denoted w_{fm}^l, b_{fm}^l is the bias parameter, * is the convolution operation, and $H_{fm}^l(.)$ is the activation function of the lth module. In Tinier-YOLO, $H_{fm}^l(.)$ is the common leaky ReLU activation. By introducing the idea of DenseNet [10], the lth fire module of Tiner-YOLO receives the feature maps of all preceding fire modules as in (4), $[x_{fm}^0, x_{fm}^1, \ldots, x_{fm}^{l-1}]$ refers to the concatenation of the feature maps produced in the fire modules.

$$x_{fm}^l = H\left(x_{fm}^{l-1} * w_{fm}^l + b_{fm}^l\right) \tag{3}$$

$$x_{fm}^l = H\left(\left[x_{fm}^0, x_{fm}^1 \ldots x_{fm}^{l-1}\right] * w_{fm}^l + b_{fm}^l\right) \tag{4}$$

AIRSPACE SYS. INC. EX2073, Page 18 (IPR2025-00629)

Comparing (3) and (4), the lth fire module with dense connection always contains feature maps from the previous $l-1$ fire modules, which can strengthen the feature propagation in the network and is expected to improve accuracy.

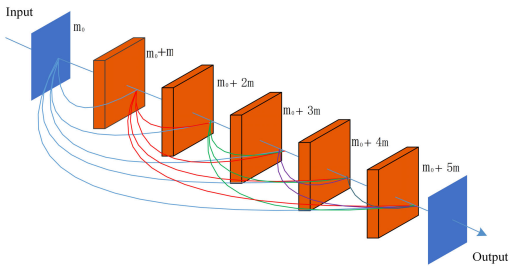


FIGURE 4. Dense connection between the fire modules.

Fig. 4 shows the dense connection between five fire modules in Tinier-YOLO. The feature maps of the first $l-1$ fire modules are concatenated and utilized as the input of the lth fire module. m_0 is the feature map of the previous convolutional layer and m is the feature map of the fire modules. Therefore the lth fire module outputs $m_0 + m(l-1)$ feature maps. The feature maps are relatively smaller that with fewer parameters and calculations.

If the dense connection is deployed in larger feature maps or connects the feature maps with different size, it will result in a huge amount of calculation, which greatly affects the real-time performance. We have tested the real-time performance of four networks with different dense connections. As shown in Table 2, the proposed dense connection in this paper has the best real-time performance with 25.1 FPS.

TABLE 2. Comparison of the real-time performance.

Connection Type	Real-time performance(FPS)
Tiny-YOLO-V2	15.5
Tiny-YOLO-V3	**21.6**
Between convolutional layers	13
Between fire modules	**25.1**

3) RETAINING FINE-GRAINED FEATURES

In Tinier-YOLO, the passthrough layer is utilized to achieve the combination of multi-scale features. Passthrough layer is added in Tinier-YOLO to acquire the upper level features of 26×26 and combine this feature with the final 13×13 output features. Tinier-YOLO combines the passthrough layer with the output of the 5th fire module and the 5th convolutional layer, which does not bring too many parameters and computations and improves the detection accuracy. The passthrough layer can fuse the underlying features of the network and also improve the ability of the model to detect small targets. At the same time, the boxes are predicted at two different scales which are 13×13 and 26×26 and the passthrough layer is utilized to take the feature maps from the previous fire module and upsample it by $2\times$ just like that in Tiny-YOLO-V3. Then two fire modules are added to process this combined feature maps and eventually a similar tensor is predicted.

4) BATCH NORMALIZATION IN Tinier-YOLO

Since the distribution of each layer's inputs in DNNs changes during training, batch normalization is proposed to address this problem and has been widely used in training DNNs that helps to use much higher learning rates [27]. Before inputting to the next layer, batch normalization takes the output of one layer and transforms its mean variance and standard deviation.

Batch normalization was introduced in YOLO series since the version of YOLO-V2. However, the computational cost has been increased 30% if batch normalization is used in every layers since means and variances of batch normalization are adjusted constantly during training [27]. The same problem also arises in Tiny-YOLO-V3. In the proposed Tinier-YOLO, fire modules have been involved with batch normalization being operated in each convolutional layer. In order to reduce the computational cost, we tried to remove batch normalization from the fire modules of Tinier-YOLO. We found that the training time for the model has been decreased after removing batch normalization in fire modules. And we also carried out the test on a total of 4952 images to evaluate the performance of Tinier-YOLO with batch normalization in every layers (Tinier-YOLO-BN) and with batch normalization in every layers except that in the fire modules (Tinier-YOLO). Table 3 gives the results of average test time, mAP, and model size. The results show that the detection accuracy has not been changed while the real-time performance and model size have slight improvement after the batch normalization is removed from the layers in the fire modules of Tinier-YOLO.

TABLE 3. Comparison between Tinier-YOLO with and without batch normalization.

	Average test time (s)	mAP	Model size (MB)
Tinier-YOLO	**195**	65.7%	**8.8**
Tinier-YOLO-BN	197	65.7%	8.9

IV. EXPERIMENTAL RESULTS

In this section, the experimental settings and experimental datasets are introduced firstly. Then the performance comparison under the metrics of mAP, BFLOP/s, model size, and FPS are carried out. Furthermore, in order to verify the improvement of Tiner-YOLO compared with the other lightweight architectures, we conduct extensive experiments to illustrate the performance comparison results.

A. EXPERIMENTAL SETTINGS

Tinier-YOLO is trained on NVIDIA Titan X in order to save the training time. The detection speed is tested on Jetson TX1, which is an embedded AI device built around the revolutionary NVIDIA Maxwell architecture with 256 CUDA

AIRSPACE SYS. INC. EX2073, Page 19 (IPR2025-00629)

TABLE 4. Comparison of configuration between Titan X and Jetson TX1.

Computing platform	GPU	Data Storage	Memory
Titan X	NVIDIA Pascal, 3584 CUDA cores	1TB	10GB 384bit GDDR5X
Jetson TX1	NVIDIA Maxwell, 256 CUDA cores	16 GB	4 GB 64 bit LPDDR4

cores delivering over 1 TeraFLOPs of performance. Jetson TX1 is suitable for embedded deep learning, computer vision, graphics, and GPU computing. Table 4 gives the configuration comparison between Jetson TX1 and TITAN X. There is only 16GB data storage of Jetson TX1, which is even smaller than normal mobile phones. The default power mode of Jetson TX1 is used for testing the detection speed.

The source codes and architecture of Tinier-YOLO is available for download here: https://github.com/jnfrancis/Tinier-YOLO.

B. EXPERIMENTAL DATASETS

The datasets of PASCAL VOC [41] and COCO [42]are used in this paper. These two datasets are usually used for image classification, object detection, and image segmentation. The dataset of PASCAL VOC includes PASCAL VOC 2007 and PASCAL VOC 2012. There are 20 kinds of objects with 29503 images. When calculating the AP values and mAP, the default value of 0.5 is used for the intersection over union (IOU). The dataset of COCO is more difficult to train than PASCAL VOC. For object detection algorithms, the performance of a model is usually more inclined on COCO. There are 80 categories of objects in COCO and the number of targets contained in each image is three times more than PASCAL VOC. The mAP value of each kind of object is the average by 10 values of IOU from 0.5 to 0.95 and mAP value is also compared on small, medium, and large size of object. The performance evaluation by multiple IOU values on COCO can better reflect the comprehensive performance of the algorithm. Table 5 shows the comparison between PASCAL VOC and COCO in detail.

TABLE 5. Detailed comparison between PASCAL VOC and COCO.

Dataset	PASCAL VOC	COCO
Number of classes	20	80
Training dataset	16551	117264
Test dataset	4952	5000
Number of ground truth boxes	52090	902435
Number of boxes per image	2.4	7.4

C. MODEL TRAINING

The darknet53.conv.74 is loaded in Tinier-YOLO before training on both datasets which is the same as that in Tiny-YOLO-V3. The darknet53.conv.74 is the pre-trained weight model based on the darknet53 [9]. The input size of Tinier-YOLO is 416*416.

The number of maximum training batches of Pascal VOC is 500K and that of COCO is 2000K, which are set according to the number of classes of each dataset. The batch size of Tinier-YOLO is 64, the subdivision is 8, the momentum is 0.9 and the decay is 0.005. In order to make the model converge much faster in the early training period, the learning rate is 0.001 and then decreased to 0.1 times of the original as the number of iteration batch reaches 400K and 450K.

D. ABLATION EXPERIMENTS FOR Tinier-YOLO

In order to show the effects of the proposed techniques on the performance of Tinier-YOLO, we carried out the ablation experiments by adding fire module, dense connection, and passthrough layer successively in the network of Tinier-YOLO. Table 6 gives the results of mAP, BFLOP/s, FPS, and model size by the ablation experiments.

As shown in Table 6, compared with Tiny-YOLO-V3, the introducing of fire module reduces the model size and BFLOP/s greatly without decreasing the detection accuracy significantly. The model size is decreased from 34.9 MB of Tiny-YOLO-V3 to 6.9 MB while keeping high detection accuracy. The employment of dense connection between the fire modules and passthrough layer makes a great contribution to the improvement in mAP without increasing too many parameters. Employing dense connection increases the mAP by 3.5% on the basis of introducing fire modules. Utilizing the passthrough layer increases mAP by about 1% on the basis of introducing dense connection between the fire modules.

TABLE 6. Ablation experiments from Tiny-YOLO-V3 to Tinier-YOLO on PASCAL VOC.

	Tiny-YOLO-V3			Tinier-YOLO
Fire Module		✓	✓	✓
Dense Connection			✓	✓
Passthrough Layer				✓
Model Size(MB)	34.9	**6.9**	8.3	8.9
BFLOP/s	5.474	**2.350**	2.552	2.563
mAP(%)	61.3	60.6	**64.8**	**65.7**
FPS	21.6	25.5	24.9	**25.1**

1) PERFORMANCE COMPARISON ON BFLOP/s AND THE MODEL SIZE

BFLOP/s is often used as a measure of the computational power required by a model. The larger the BFLOP/s, the higher the requirements for the device. As shown in Table 7, BFLOP/s of Tinier-YOLO is only 2.563, which is less than the half of Tiny-YOLO-V3 and is suitable for constrained environment. The model size of Tinier-YOLO is almost 4× smaller than Tiny-YOLO-V3, 7× smaller than Tiny-YOLO-V2.

2) COMPARISON ON REAL-TIME PERFORMANCE

The reference point of measuring real-time performance is FPS and depends on the resolution of the test video or

AIRSPACE SYS. INC. EX2073, Page 20 (IPR2025-00629)

TABLE 7. Performance of BFLOP/s and model size.

	YOLO-V1	YOLO-V2	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
Size	1.1G	202.7MB	63.4MB	34.9MB	**8.9MB**
BFLOP/s	40.19	34.9	6.97	5.474	**2.563**

TABLE 8. Comparison of the real-time performance.

	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
Average test time(s)	320.2	229.0	**196.9**
FPS	15.5	21.6	**25.1**

the image. With the support of Jetson TX1, we measured the total time for testing 4952 images in the VOC 2007 dataset. The average value of testing time by 30 times is shown in Table 8. It can be seen that the total time required for the Tinier-YOLO to detect about 5,000 pictures is less than 200 seconds. Tinier-YOLO performs faster than Tiny-YOLO-V3. Tiny-YOLO-V2 takes more than 320 seconds to complete the detection. Tinier-YOLO can detect objects with 25.1 FPS. The real-time performance is increased by 16.2% compared with Tiny-YOLO-V3 and increased by 39.4% compared with Tiny-YOLO-V2.

3) PERFORMANCE COMPARISON ON mAP

mAP is usually used to measure the detection accuracy. On the VOC 2007 dataset, the AP values of the three compared methods are shown in Table 9. From the average AP values, it can be seen that Tinier-YOLO improved the detection accuracy by 4.5% compared with Tiny-YOLO-V3, and nearly 8.6% compared with Tiny-YOLO-V2. The AP values of all the 20 classes are better than Tiny-YOLO-V2. Compared with Tiny-YOLO-V3, except for the class of aeroplane, the AP values of the other 19 classes are all better than Tiny-YOLO-V3, which demonstrates the effectiveness of Tiner-YOLO.

TABLE 9. The AP values of three methods on PASCAL VOC dataset.

	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
aeroplane	64.0	**69.9**	67.8
bicycle	74.7	75.2	**77.6**
bird	48.7	44.6	**51.5**
boat	41.8	50.4	**53.8**
bottle	18.2	33.0	**43.8**
bus	70.1	73.0	**76.8**
car	69.4	77.5	**80.9**
cat	72.7	68.4	**77.8**
chair	33.4	38.6	**44.7**
cow	53.9	60.0	**68.8**
diningtable	58.5	59.2	**61.2**
dog	63.6	61.2	**67.9**
horse	73.8	75.6	**80.3**
motorbike	71.4	75.8	**76.3**
person	61.6	71.6	**75.4**
pottedplant	25.6	28.4	**38.8**
sheep	56.1	64.0	**66.0**
sofa	52.8	58.8	**62.2**
train	72.9	75.1	**75.9**
tvmonitor	60.1	65.0	**66.8**
avg	57.1	61.3	**65.7**

On the COCO dataset, the mAP results of Tiny-YOLO-V2, Tiny-YOLO-V3, and Tinier-YOLO are shown in Table 10.

TABLE 10. Performance of mAP on the COCO dataset.

IOU	area	maxDets	Tiny-YOLO-V2	Tiny-YOLO-V3	**Tinier-YOLO**
mAP(%) 0.50:0.95	all	100	9.9	15.3	**17.0**
mAP(%) 0.50	all	100	23.7	33.1	**34.0**
mAP(%) 0.75	all	100	6.8	12.4	**15.7**
mAP(%) 0.50:0.95	small	100	7.0	4.4	**4.8**
mAP(%) 0.50:0.95	medium	100	7.8	15.2	**17.3**
mAP(%) 0.50:0.95	large	100	20.1	25.1	**26.8**
AR(%) 0.50:0.95	all	1	12.4	16.7	**18.5**
AR(%) 0.50:0.95	all	10	19.4	25.9	**28.3**
AR(%) 0.50:0.95	all	100	21.1	27.5	**30.0**
AR(%) 0.50:0.95	small	100	2.3	7.7	**8.4**
AR(%) 0.50:0.95	medium	100	20.4	30.3	**33.9**
AR(%) 0.50:0.95	large	100	41.1	44.4	**47.2**

The mAP value under IOU = 0.5 of Tinier-YOLO is almost 1% higher than that of Tiny-YOLO-V3. In case of more strict tests under IOU = 0.75 and IOU from 0.50 to 0.95, the mAP values of Tinier-YOLO are 1.7% and 3.3% higher than Tiny-YOLO-V3. The Average Recall (AR) of Tinier-YOLO is also better than Tiny-YOLO-V3. Table 10 also indicates that Tinier-YOLO makes more improvements in detecting medium and large objects than detecting small objects.

4) PERFORMANCE COMPARISON BETWEEN Tinier-YOLO AND OTHER LIGHTWEIGHT MODELS

In order to further evaluate the improvement of Tiner-YOLO comparing to the other state-of-the-art lightweight architectures, we take SqueezeNet SSD and MobileNet SSD [43] into comparison. The models of SqueezeNet SSD and MobileNet SSD are trained on PASCAL VOC 2007 + 2012 training dataset and tested on PASCAL VOC 2007 test dataset. The training process is completed on NVIDIA Titan X and the test is completed on Jetson TX1, which uses the same experimental environment as Tinier-YOLO. The maximum number of iterations of SqueezeNet SSD is set as 200K and that of MobileNet SSD is set as 120K.

TABLE 11. Comparison between Tiner-YOLO and other lightweight models.

Model	Model Size(MB)	BFLOP/s	mAP(%)	Average test time(ms)
SqueezeNet SSD	22.2	4.499	64.3	595.7
MobileNet SSD	23.3	4.288	**72.4**	483.7
Tinier-YOLO	**8.9**	**2.563**	65.7	**135.2**

Table 11 summarizes the comparison results of the three models. As can be seen from Table 11, Tinier-YOLO achieves much better results in model size, BFLOP/s, and average test time than MobileNet SSD and SqueezeNet SSD. The model size of Tinier-YOLO is more than 2× smaller than SqueezeNet SSD and MobileNet SSD. The BFLOP/s value of Tinier-YOLO is about 2× smaller than the other two models. For the average test time, it is calculated by the average time of 30 images processed by the three models. From Table 11, Tiner-YOLO takes 135.2ms to detect an image, while MobileNet SSD uses 483.7ms and SqueezeNet SSD uses 595.7ms, which are more time-consuming than

AIRSPACE SYS. INC. EX2073, Page 21 (IPR2025-00629)

Tinier-YOLO. As mAP is concerned, Tiner-YOLO achieves the result of 65.7% mAP, which is better than SqueezeNet SSD and worse than MobileNet SSD. Considering the overall performance, Tiner-YOLO reduces more than 50% of the model size and 40% of the BFLOP/s of computing with a faster runtime speed, which indicates that the proposed Tiner-YOLO is a competitive model for the constrained scenario.

V. CONCLUSION

In this paper, we have proposed Tinier-YOLO which is a real-time object detection method for constrained environments. Tinier-YOLO is designed by introducing the fire module of SqueezeNet into Tiny-YOLO-V3 at first in order to reduce the model size. And then the number of fire models and their positions in the network architecture have been studied. The connectivity method between fire modules in Tinier-YOLO is realized by using dense connections of DenseNet. The dense connections in Tinier-YOLO helped to improve the detection accuracy and real-time performance since the feature propagation is strengthened and the maximum information flow in the network is ensured. By incorporating the passthrough layer in Tinier-YOLO, the detection accuracy has been further improved by merging feature maps from the front layers to get fine-grained features. The computational cost of the network has been reduced by removing the batch normalization from the fire modules of Tinier-YOLO. The experiments on PASCAL VOC and COCO demonstrate Tinier-YOLO is more efficient compared with Tiny-YOLO-V3. Comparing with lightweight models like SqueezeNet SSD and MobileNet SSD, Tinier-YOLO also shows its competitive performance in terms of model size, BFLOP/s, and average test time. The results demonstrate the efficacy of Tinier-YOLO for constrained environments. In future, we look forward to further optimizing the performance of Tinier-YOLO, especially for the COCO dataset.

REFERENCES

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

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2014, pp. 346–361.

[3] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[5] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 379–387.

[6] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *Proc. CVPR*, 2017, vol. 1, no. 2, pp. 2117–2125.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," 2017, *arXiv:1612.08242*. [Online]. Available: https://arxiv.org/abs/1612.08242

[9] J. Redmon and A. Farhadi, "YOLOV3: An incremental improvement," 2018, *arXiv:1804.02767*. [Online]. Available: https://arxiv.org/abs/1804.02767

[10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. CVPR*, 2017, vol. 1, no. 2, pp. 4700–4708.

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2009.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[14] J. Jeong, H. Park, and N. Kwak, "Enhancement of SSD by concatenating feature maps for object detection," 2017, *arXiv:1705.09587*. [Online]. Available: https://arxiv.org/abs/1705.09587

[15] C. Fu, W. Liu, A. Ranga, A. Tyagi, and A. Berg, "DSSD: Deconvolutional single shot detector," 2018, *arXiv:1701.06659*. [Online]. Available: https://arxiv.org/abs/1701.06659

[16] Z. Li and F. Zhou, "FSSD: Feature fusion single shot multibox detector," 2017, *arXiv:1712.00960*. [Online]. Available: https://arxiv.org/abs/1712.00960

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published.

[18] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," 2019, *arXiv:1903.00621*. [Online]. Available: https://arxiv.org/abs/1903.00621

[19] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint triplets for object detection," 2019, *arXiv:1904.08189*. [Online]. Available: https://arxiv.org/abs/1904.08189

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: https://arxiv.org/abs/1510.00149

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[22] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," 2016, *arXiv:1602.02830*. [Online]. Available: https://arxiv.org/abs/1602.02830

[23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[24] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4133–4141.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[26] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: https://arxiv.org/abs/1312.4400

[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: https://arxiv.org/abs/1502.03167

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.

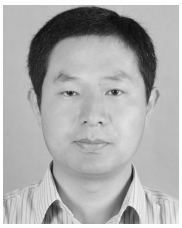
[29] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-V4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI*, vol. 4, 2017, p. 12.

[30] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.

[31] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: https://arxiv.org/abs/1602.07360

AIRSPACE SYS. INC. EX2073, Page 22 (IPR2025-00629)

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.
[33] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 116–131.
[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861
[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," 2018, *arXiv:1801.04381*. [Online]. Available: https://arxiv.org/abs/1801.04381
[36] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, and V. Vasudevan, "Searching for mobilenetv3," *arXiv preprint arXiv:1905.02244*, 2019.
[37] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 2820–2828.
[38] Y. N. Dauphin and Y. Bengio, "Big neural networks waste capacity," Jan. 2013, *arXiv:1301.3583*. [Online]. Available: https://arxiv.org/abs/1301.3583
[39] L. J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2013.
[40] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, "Exponential expressivity in deep neural networks through transient chaos," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Dec. 2016.
[41] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.
[42] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2014, pp. 740–755.
[43] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade–offs for modern convolutional object detectors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 7310–7311.



WEI FANG (M'15) received the Ph.D. degree in information technology and engineering of light industry from Jiangnan University, Wuxi, China, in 2008. He was a Visiting Scholar with the CEACIA, University of Birmingham, Birmingham, U.K., from April 2013 to April 2014. He is currently a Professor with the Department of Computer Science, School of IoT Engineering with Jiangnan University. He has published more than 50 scientific articles in journals and international conferences. His current research interests involve the evolutionary computation, swarm intelligence, multiobjective optimization, and large-scale global optimization. He serves as an Editorial Board Member of the *International Journal of Swarm Intelligence Research* and the *International Journal of Computing Science and Mathematics*.



LIN WANG is currently pursuing the M.S. degree with the School of IoT Engineering, Jiangnan University, Wuxi, China. Her research interests include deep learning and computer vision.



PEIMING REN received the master's degree in computer science and technology from the School of IoT Engineering, Jiangnan University, in 2019. His research interests include issues related to artificial intelligence and pattern recognition, computer vision, and machine learning.

AIRSPACE SYS. INC. EX2073, Page 23 (IPR2025-00629)