International Journal of Informatics and Applied Mathematics
e-ISSN:2667-6990 Vol. 2, No. 2, 28-44

A Survey on Lightweight CNN-Based Object Detection Algorithms for Platforms with Limited Computational Resources

Bouguettaya Abdelmalek, Kechida Ahmed, and Taberkit Mohammed Amine

Research Center in Industrial Technologies CRTI, P.O. Box 64, Cheraga. Algeria
{a.bouguettaya,a.kechida,a.taberkit}@crti.dz

Abstract. Autonomous drones must be able to identify the existence of one or more objects of interest in a complex environment with high accuracy and speed to fly around safely. Most existing object detection techniques, based on traditional machine learning algorithms, can't offer acceptable performance in complicated environments. Deep Convolutional Neural Networks (CNNs) provide us such ability with high performance. Today, deep CNN-based object detection algorithms are more and more used in Artificial Intelligence (AI) applications. However, it still very difficult to deploy large CNNs architectures on small devices with limited hardware resources, because they consist of millions of parameters, which make them computationally very exhausting. Lightweight CNN architectures are proposed as a solution to make the deployment of deep neural networks on small devices feasible. This paper focuses on reviewing recent used lightweight CNN architectures that can be implemented on embedded targets to improve the object detection performance for small devices-based systems, like drones. We need to select fast and lightweight CNN models to use them on drone platforms. The purpose of this reviewing is to choose the most accurate and fastest algorithm to implement it on our drones.

Keywords: Computer Vision · Deep Learning · Object Detection · Convolutional Neural Network · Lightweight CNN.

1 Introduction

Today, with the emergence of Internet of Things and embedded systems, artificial intelligence and computer vision has entered our lives. Our smartphones use it to improve the quality of our photos, autonomous drones to understand their environment. In general, our embedded systems analyze more and more images. Most of the applied deep learning operations are made on very powerful workstations or servers. This is because deep neural networks perform convolutions, which are very expensive operations in the processing and memory. The classification of images in on-board embedded systems is, therefore, a major challenge due to material constraints.

In the last few years, deep learning has shown accuracy in many applications. Thanks to Moore's law [1], we are able to implement high-performance processors in a single small chip, which gives us the ability to implement very efficient deep learning algorithms in these little chips in order to build a fully autonomous drone that could navigate without the intervention of humans or making smartphones with high capabilities. Also, we have many efficient software tools at our disposal, like Keras, Tensorflow, and Theano. These hardware and software give us the ability to rapidly construct deep learning architectures in a fraction of the time. While they took a very long time to build such architectures before the advent of the recent hardwares and softwares tools.

Computer vision is one of the topics that is advancing rapidly thanks to deep learning. Today, deep learning-based computer vision is helping self-driving cars and autonomous drones by figuring out where are other objects (pedestrians, cars, traffic signs) to avoid them. It is making face recognition much better than ever before. Rapid advances in computer vision are enabling us to build new applications that were impossible a few years ago. Thanks to CNN [2], we can use large images instead of stacking with small images.

The purpose of computer vision is allowing computers or robotic systems to analyze, process and understand the content of digital images acquired with cameras so that they can decide how to act. For a while, we needed to apply a hand-engineered algorithm, where a hand-defined set of rules and algorithms are applied to extract features from an image. However, the Convolutional neural network is an end to end model that gives us the possibility to skip the feature extraction step. This step is automatically learned from the training process. Researchers needed to develop a deep neural network to detect objects. Several studies have been made to tackle the object detection problem using CNN.

A large number of CNN architectures have been developed to achieve high accuracy on many datasets, like ImageNet. In such competitions computing power is not limited, where they use very powerful GPUs. However, we may want to run our model on an old laptop without GPU, on our smartphone, or even on autopilot for drones. In this paper, we will present an overview of the most common and recent lightweight CNN-based approaches, which were used for embedded systems implementation.

The rest of the paper is organized as follows. Section 2 reviews the prior works of deep CNN-based object detection algorithms. In section 3, we introduce an overview of deep learning and different deep CNN architectures. In section 4, we present the most used and accurate lightweight CNN architectures. Finally, we conclude this paper.

2 Object Detection Related Works

There were many different techniques for detecting objects applied on datasets like PASCAL VOC, MS COCO, ImageNet. Paul Viola and Michael Jones came up, in 2001, with a very effective object detection method [3], and in 2002, it was improved by Rainer Lienhart and Jochen Maydt [4]. Viola-Jones still one of the most powerful algorithms for computer vision and real-time object detection. Other methods, like HOG+SVM [5] and DPM [6], present good accuracy in the mentioned datasets. It is slowly being surpassed by deep learning-based CNN algorithms in terms of accuracy.

Girshick et al. [7], proposed Region-based Convolutional Neural Networks (R-CNN), which achieve impressive object detection accuracy, over traditional methods. However, the R-CNN is very slow, where detection time takes 47s per image. Besides, the extracted features need a huge storage memory, hundreds of Gigabytes [7]. Faster R-CNN [8] and Faster R-CNN [9] have come to enhance the accuracy and speed of R-CNN architecture. However, they still slow achieving only 7 fps (frame per second), which is not suitable for real-time object detection.

Single-stage detectors are significantly faster than two stages of detectors (region-based methods), and give us a real-time performance. YOLO [10, 11, 12] and SSD [13] are the most used algorithms for real-time object detection, where many works are based on these two architectures. They are prerequisites for any self-driving cars and autonomous drones. If they are going to be driving or fly around safely, they need to be able to recognize all the objects around them and need to do it fast. It is not just an interesting computer science problem, but it is now a safety requirement. Table 1 presents a comparison between different object detection methods based on CNN architectures.

All the previously mentioned object detection techniques, in table 1, are based on deep CNN architectures, like AlexNet [14], VGGnet [15], GoogLeNet [16], and ResNet [17]. All of these architectures are computationally exhausting, which makes them not suitable for embedded systems implementation.

Table 1. Comparison of accuracy and speed on PASCAL VOC 2007.

	mAP (%)	FPS	Real-time
Fast R-CNN	70.0	0.5	No
Faster R-CNN(VGG16)	73.2	7	No
Faster R-CNN(ZF)	62.1	18	No
Faster R-CNN(ResNet-101)	76.4	5	No
Fast YOLO	52.7	155	Yes
YOLO	63.4	45	Yes
YOLO (VGG16)	66.4	21	No
YOLOv2	78.6	40	Yes
SSD300	74.3	59	Yes
SSD512	76.8	19	No

3 Deep Learning and CNN Overview

Deep learning is an exciting branch of machine learning, which is, in turn, a subfield of artificial intelligence. It uses a huge amount of data to train machines how to do things only humans were capable of before. Solving the problem of perception, recognizing what's in an image, helping self-driving cars explore its environment and interact with it, are some of the most exciting and challenging topics [18, 14].

Deep Learning has emerged as a central tool to solve perception problems in the last decade. It's the state of the art on everything having to do with computer vision and speech recognition. Increasingly, people are finding that deep learning is a much better tool to solve problems, like understanding natural language [19], detecting objects in a scene [20], understanding documents. In this paper, we are interested in studying a special type of deep learning algorithms, which is Convolutional Neural Networks (CNN) [2]. It is very useful for computer vision and image classification. Deep CNN architectures are computationally very exhausting. So, we need to build lightweight CNN architectures for edge computing. Drones are one of the fields where we can implement lightweight CNN algorithms on small devices with limited hardware resources to make them more intelligent.

Convolutional Neural Network does one basic thing; image classification. We can turn CNN into an object detection system that not only classifies images but can locate each object in it and predict its label. In traditional feedforward neural networks, each neuron in a layer "i" is connected to every neuron in layers "i-1" and "i+1". We call it a Fully Connected Neural Network (FCNN). However, in CNN, we dont use fully connected layers until the last layer(s) in the network [21]. CNN presents many advantages over FCNN in high-dimensional data analysis, like images and videos.

A nonlinear activation function is applied to the output of each convolution layer. This process continues along with a mixture of convolution and pooling layers to extract features, until we reach the end of the network and apply one or two fully connected layers, where we can obtain our final output classifications.

3.1 CNN Building Blocks

CNN is composed of four fundamental layers (building blocks). In this section, we focus on the different layer types associated with CNNs. Typically, CNNs are composed of many convolutional layers, where each layer followed by a nonlinear activation function and pooling layers that are followed by one or more fully connected layers at the end.

The Convolutional Layer The convolutional layer is considered as the most important component of CNN architectures. It consists of set of filters (kernels), which are convolved with the input image to generate an output feature map (Fig. 1). The weights of each filter are learned during the training of CNN.

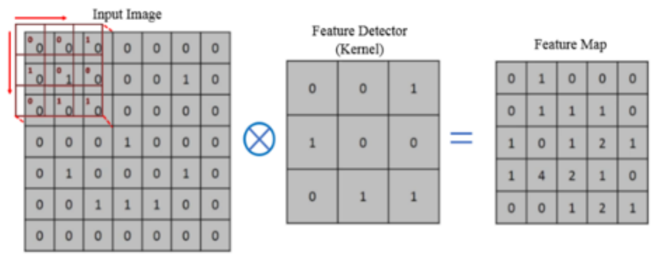


Fig. 1. The convolution operation.

Figure 2 presents a convolutional layer, which looks for six different features (edges, lines, corners). In this case, our convolutional layer will have six (3*3*3) filters, as shown in figure 2. Each filter looking for a particular pattern on the image (edges, corners, lines).

The Activation Layer The convolution by itself is a linear operation. However, a non-linear layer at the end of each convolutional layer is added in order to avoid the encounter with the same problem of the linear classifiers. The Rectified Linear Unit (ReLU) (Fig. 3) is one of the most used activation functions. AlexNet was the first architecture to implement ReLU as an activation function.

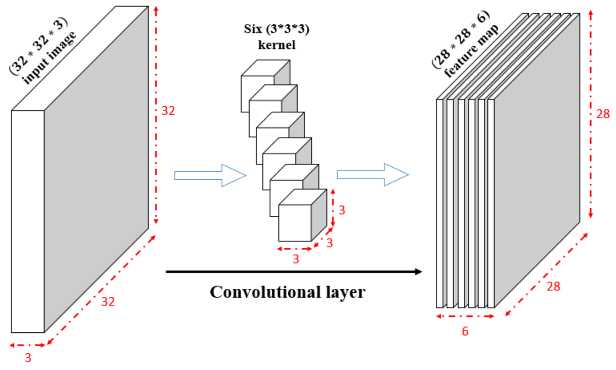


Fig. 2. Convolutional layer with six (3*3*3) filters.

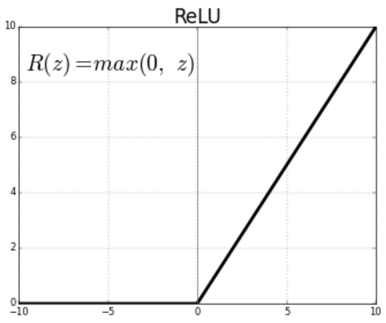


Fig. 3. Rectified Linear Unit activation function.

Pooling Layer The pooling layer, or subsampling layer, is for downsampling the feature map of the previous layer. Generally, it is applied in two forms: max-pooling and average pooling. It is used to reduce the size of an image by discarding some information to speed up the computation, but it preserves the information about the location of the good matches with the filters. As shown in figure 4, max-pooling preserves maximum value in a patch and stores it in the new image. A pooling layer operates on the heights and the widths of the data tensor, but not on the depth. In the case of average pooling, the same thing as maximum pooling, only instead of taking the maximum value we take the average of all the values and put it in the corresponding position of the output matrix. These days, max pooling is used much more than average pooling.

Fully Connection Layer Fully connected layers are feedforward neural networks, which are the same as classical artificial neural networks and perform the same mathematical operations. The output of the final pooling layer is flattened

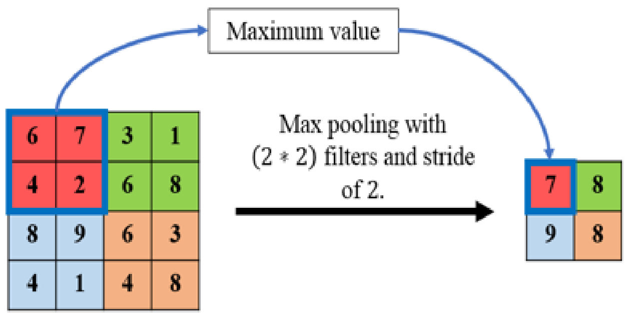


Fig. 4. Max pooling operation.

and then fed to the input of the fully connected layer, which does the classification and produces the final output probabilities for each class using a softmax activation function.

The dimensions of its output is [1* 1* N], where N is the number of classes we are evaluating (for example N = 10 for CIFAR-10 dataset). Each neuron in this layer is connected to every neuron in the previous and next layer.

3.2 Different Deep CNN Architectures

In this section, we will present four of the most powerful CNNs, which laid the foundation for todays computer vision achievements.

AlexNet In 2012, Krizhevsky et al. [14] propose AlexNet, which is one of the most powerful models for object detection. It won easily the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) competition [22]. It consists of five convolutional layers and three fully connected layers (Fig. 5). It was the first architecture that uses the ReLU activation function instead of tanh function.

VGGnet VGGnet [15] was the winner of the ILSVRC 2014 competition for localization and classification. This network has two famous architectures: VGG-16 and VGG-19. The VGG-16 is the most used architecture due to its simplicity. It contains 13 convolutional layers and 3 fully connected layers, which make them 16 layers in total (Fig. 6).

GoogLeNet GoogLeNet [16] is another CNN-based architecture, which consists of 22 layers (Fig. 7). In fact, it has more than 50 convolutional layers distributed inside the inception modules [21]. However, it reduces the number of parameters of the network from 60 million (in AlexNet) to only 4 million, which makes it less sensitive to overfitting and allows it to be deeper.

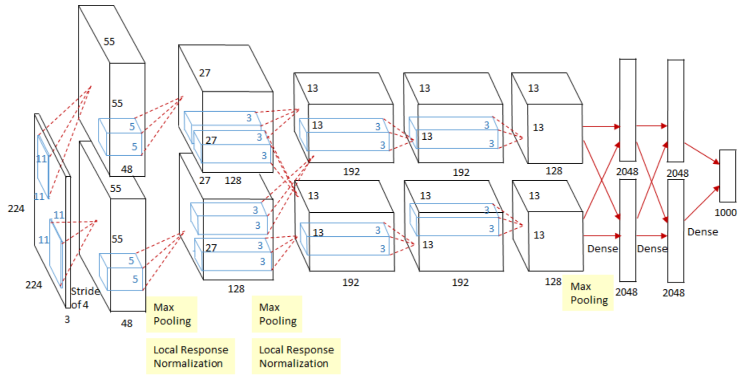


Fig. 5. AlexNet architecture.

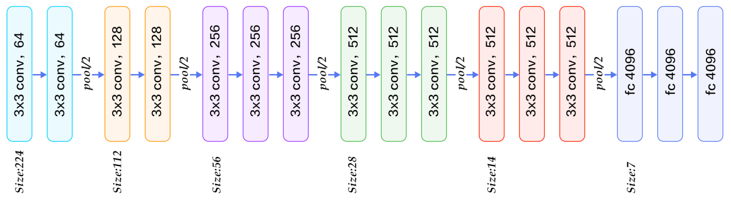


Fig. 6. VGG-16 architecture.

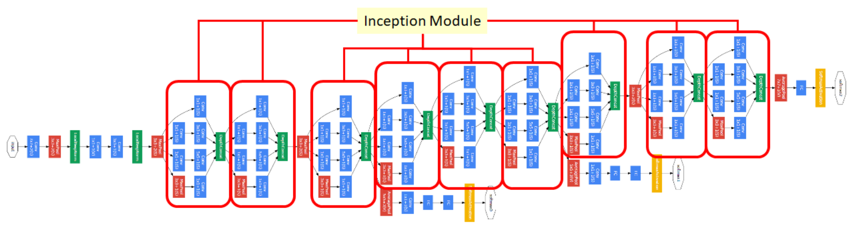


Fig. 7. GoogLeNet architecture.

ResNet Residual Neural Network (ResNet) [17] CNN architecture (Fig. 8) won the first places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation at the 2015 ILSVRC and COCO competitions. It introduced a new architecture with skip connections and features heavy batch normalization, which makes it able to train a 152 layers neural network while still having lower complexity than VGGNet.

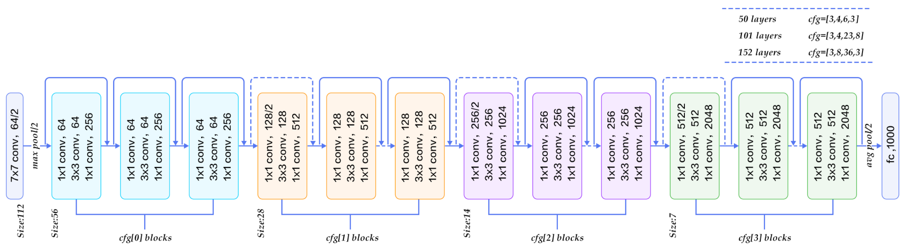


Fig. 8. ResNet architecture.

All the discussed CNN architectures, in this section, need large memory and powerful computational resources. For this reason, deep CNN architectures are unable to run on small devices with low hardware resources, like drone autopilots. Many accurate lightweight CNN models are proposed to work with high speed on embedded systems as we will present in the next section.

4 On-board Lightweight CNNs Architectures

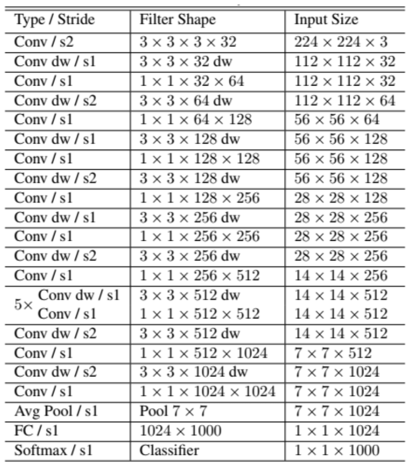
In the case of object detection based on high-performance computers, the system (drone, car) collects data via its cameras and different sensors and transmits them to a desktop computer for analysis. This gives machines the ability to save power by off-loading compute intensive operations. However, the wireless transmission takes a long time in sending the collected data, which means an additional cost added to the latency of the system. The requirement of high computational resources for video processing poses a challenge in mapping deep learning-based algorithms on low-cost and low-power computing platforms. In a large number of applications such as robotics, self-driving car, autonomous drones and augmented reality, the recognition tasks need to be carried out in a timely fashion on a computationally limited platform [23].

In order to solve these problems, developing CNNs that are suitable for on-board real-time object detection is important to reduce model parameters and accelerate their calculations. For that purpose, lightweight versions of the CNNs architectures are used on limited hardware.

All the previous methods are used with high-performance computers. Lightweight versions are used for embedded systems, for example on the drone itself. The battery life and flight time could be increased using lightweight models. An additional 0.5 to 1 W power is required to operate the cooling system for each watt of power dissipated in computing equipment [24]. Moreover, a low-power computing system can reduce thermal problems and cooling requirements, which is very important in the field of autonomous drones.

4.1 MobileNet

MobileNet [23] is an efficient CNN architecture for mobile and embedded vision systems. It splits the convolution into a depthwise separable convolution followed by a pointwise convolution to build a lightweight deep neural network (Fig. 9). Furthermore, it introduces two simple hyperparameters that give us the possibility to build small and low latency models that can be easily matched to the design requirements for mobile and embedded vision applications. One of the hyperparameters is the width multiplier that allows us to thin the number of channels, while the second hyperparameter is the resolution multiplier that reduces the spatial dimensions of the feature maps. MobileNet is not usually accurate as the bigger CNN architectures. However, MobileNet shines in the resource/accuracy trade-off. It gives high accuracy only with limited resources.



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig. 9. MobileNet architecture.

MobileNet V2 [25] is an updated version of MobilNet V1, which makes it more efficient and powerful in terms of accuracy and speed. Table 2 shows a comparison between the two versions of MobileNets architectures, which present that V2 is twice faster than V1 and slightly more accurate. MACs are multiply-accumulate operations, which measure the number of needed calculations in order to perform inference on a single (224*224) RGB image.

MobileNet is a very fast architecture, which is suitable for real-time applications. In [26], the authors present that MobileNet (width multiplier = 0.25 and resolution multiplier = 0.714) achieves 28.1 fps on Nvidia Jetson TX2, 31.5 fps on Intel Core i5-6200U CPU and 164 fps on K40 Desktop GPU.

In [23], the authors proposed MobileNet-SSD, which is comprised depthwise separable convolutions. They used MobileNet as a backbone CNN architecture instead of VGG-16. It achieves a significant detection accuracy on COCO dataset.

The authors in [26] propose SSDLite, which is based on MobileNet V2 as building CNN. SSDLite based on MobileNetV2 outperforms YOLOv2 on COCO dataset with 20x more efficient and 10x smaller.

Table 2. Comparison of accuracy and speed of MobileNets V1 and V2.

	MACs (million)	# Parameters (million)	Top 1 accuracy	Top 5 accuracy
MobileNet V1	569	4.24	70.6	89.5
MobileNet V2	300	3.47	71.8	91.0

4.2 SqueezeNet

SqueezeNet [27] is a small CNN architecture that achieves the accuracy of AlexNet CNN on ImageNet dataset with 50x fewer parameters (Tab. 3). SqueezeNet can be 500x smaller than AlexNet using deep compression technique [28].

Table 3. Comparison of accuracy and reduction in model size between AlexNet and SqueezeNet.

	Reduction in model size	Top 1 ImageNet accuracy	Top 5 ImageNet accuracy
AlexNet	1x	57.2	80.3
SqueezeNet	50x	57.5	80.3

SqueezeNet architecture is based on three strategies: - using (1*1) filters instead of (3*3) filters, since a (1*1) filter has 9x fewer parameters than a (3*3) filter; - decreases the number of input channels to 3x3 filters using squeeze layers; and - downsample late to keep a big feature map.

Fire module is the building brick of SqueezeNet architecture, which consists of two layers: a squeeze convolution layer, which has only (1*1) filters), and an expand layer that has a mix of (1*1) and (3*3) convolution filters (Fig. 10).

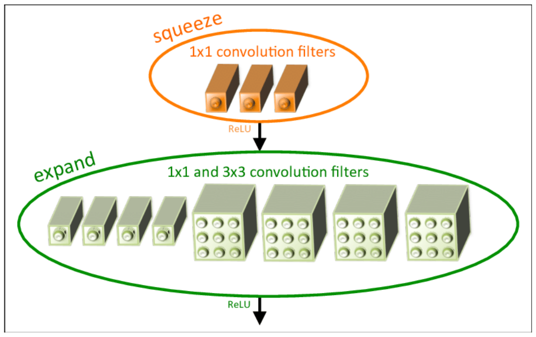


Fig. 10. Organization of convolution filters in the Fire module.

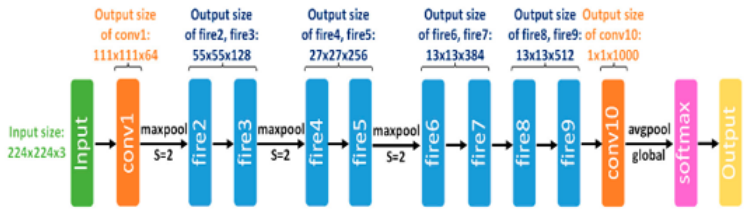

Input size: 224x224x3
Input
conv1
Output size of conv1: 111x111x64
maxpool S=2
fire2
fire3
Output size of fire2, fire3: 55x55x128
maxpool S=2
fire4
fire5
Output size of fire4, fire5: 27x27x256
maxpool S=2
fire6
fire7
Output size of fire6, fire7: 13x13x384
fire8
fire9
Output size of fire8, fire9: 13x13x512
conv10
Output size of conv10: 1x1x1000
avgpool global
softmax
Output

Fig. 11. SqueezeNet architecture.

As shown in figure 11, SqueezeNet begins with a standalone convolution layer (conv1), followed by 8 Fire modules (fire2 to fire9), ending with a final conv layer (conv10).

Wu et al. [29], proposed SqueezeDet, which is inspired by YOLO using SqueezeNet as a backbone network.

4.3 ShuffleNet

ShuffleNet [30] is a computation-efficient lightweight CNN architecture for mobile devices with limited computing power. It provides better performance than MobileNet on the tasks of ImageNet classification and COCO detection. The overall ShuffleNet architecture is shown in figure 12. It is composed of a stack of ShuffleNet units grouped into three stages. The first building block in each stage is applied with stride = 2. The output channels are the same in each stage, but doubled for the next one.

ShuffleNet architecture utilizes two new operations in order to reduce computation cost while maintaining accuracy. These operations are pointwise group convolution and channel shuffle. The channel shuffle operation permits to divide the channels in each group into many subgroups, then feed each group in the next layer with different subgroups. The authors of [30] generalize the concept of group convolution, used in AlexNet, and the separable convolution, which used in MobileNet architecture.

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

Fig. 12. ShuffleNet architecture.

Figure 13 presented the ShuffleNet units, which are specially designed for small networks. Figure 12 (a) is a bottleneck unit with depthwise convolution (3*3 DWConv). Figure 12 (b) is a ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle, which recover the channel dimension to match the shortcut path. Figure 12 (c) is a ShuffleNet unit with stride = 2.

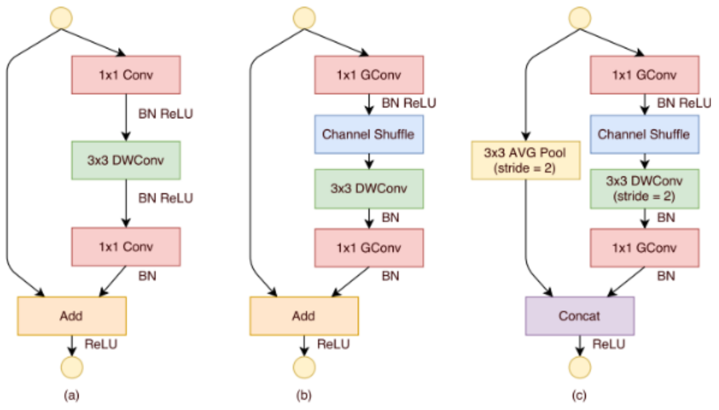


Fig. 13. ShuffleNetV1 units.

ShuffleNet achieves 13x actual speedup over AlexNet architecture, on an ARM-based mobile device, while maintaining comparable accuracy.

4.4 PeleeNet

PelleNet [31] is an efficient architecture for embedded platforms. Figure 14 shows the framework of PeleeNet, which is consists of a stem block and four stages of feature extractor. Except for the last layer in each stage, which is an average pooling layer with stride 2.

PeleeNet achieves better accuracy and 1.8x faster over the two versions of MobileNet, on ImageNet ILSVRC 2012 using Nvidia Jetson TX2. Meanwhile, the PeleeNet model size is smaller than MobileNet by 66%.

Stage		**Layer**	**Output Shape**
Input			224 x 224 x 3
Stage 0	Stem Block		56 x 56 x 32
Stage 1	Dense Block	DenseLayer **x 3**	28 x 28 x 128
	Transition Layer	1 x 1 conv, stride 1	
		2 x 2 average pool, stride 2	
Stage 2	Dense Block	DenseLayer **x 4**	14 x 14 x 256
	Transition Layer	1 x 1 conv, stride 1	
		2 x 2 average pool, stride 2	
Stage 3	Dense Block	DenseLayer **x 8**	7 x 7 x 512
	Transition Layer	1 x 1 conv, stride 1	
		2 x 2 average pool, stride 2	
Stage 4	Dense Block	DenseLayer **x 6**	7 x 7 x 704
	Transition Layer	1 x 1 conv, stride 1	
Classification Layer		7 x 7 global average pool	1 x 1 x 704
		1000D fully-connecte,softmax	

Fig. 14. PeleeNet architecture.

MACs cannot replace the speed test on real devices, because of many other factors that may influence the actual time cost, like hardware optimization, I/O. PeleeNet achieves higher accuracy and speed than MobileNet and MobileNetV2 on Nvidia Jetson TX2 (Tab. 4).

In [31], the authors combined PeleeNet with SSD for object detection. As shown in Table 5, the object detector based on PeleeNet architecture presents higher accuracy than the other architectures.

SSD + PeleeNet of [30] achieves 76.4% mAP on VOC07 and 22.4 mAP on MS COCO dataset, achieving 23.6 FPS on iPhone 8 and 125 FPS on NVIDIA Jetson TX2. It is 13.6x lower computational than YOLOv2 and 11.3 smaller. Also, it provides higher accuracy on COCO dataset.

Table 4. Comparison of deferent lightweight CNN architectures performance tested on Nvidia Jetson TX2.

	MACs (million) 224*224	# parameters (million)	Top 1 accuracy on ILSVRC 2012 224*224	Speed (img/s)		
				224*224	320*320	640*640
1.0 MobileNet V1	569	4.24	70.6	136.2	75.7	22.4
1.0 MobileNet V2	300	3.47	72.0	123.1	68.8	21.6
ShuffleNet 2x (g = 3)	524	5.2	73.7	110	65.3	19.8
PeleeNet	508	2.8	72.6	240.3	129.1	37.2

Table 5. Comparison of deferent lightweight CNN-based object detectors performance on PASCAL VOC 07 + 12.

	Input dimension	MACs (million)	Data	mAP (%)
Tiny-YOLO v2	416*416	3490	07 + 12	57.1
SSD + MobileNet	300*300	1150	07 + 12	68.0
SSD + PeleeNet	304*304	1210	07 + 12	70.9
SSD + MobileNet	300*300	1150	07+12+COCO	72.7
SSD + PeleeNet	304*304	1210	07+12+COCO	76.4

5 Conclusion

We saw that deep learning-based object detection algorithms achieve high accuracy and speed for real-time issues. Drones, cars, and robots need accurate and fast object detection methods that are suitable for its low-power and low-processing. Lightweight CNN architectures give us the possibility to execute these accurate algorithms on limited hardware resources embedded systems.

In this paper, we presented a survey of lightweight CNNs architectures for accelerating neural-network applications. We discussed both hardware-level and CNN-level optimizations and showed the speed and accuracy of each one of them on limited resources hardware, like iPhone 8 and Nvidia Jetson TX2. This reviewing will be useful for researchers and practitioners in the area of deep learning, computer vision and embedded system.

As future work, we want to implement different lightweight CNNs-based deep learning algorithms on small devices to control our drones.

References

1. Moore, G. E.: Cramming More Components onto Integrated Circuits. In proc. of the IEEE, Reprinted from Electronics, Volume 86, Issue 1, Pages 82–85 (1998). https://doi.org/10.1109/JPROC.1998.658762
2. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In proc. of the IEEE, Volume 86, Issue 11, Pages 2278–2324 (1998). https://doi.org/10.1109/5.726791
3. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Volume 1, Pages 511–518 (2001). https://doi.org/10.1109/CVPR.2001.990517
4. Lienhart, R., Maydt, J.: An Extended Set of Haar-like Features for Rapid Object Detection. In proc. in International Conference on Image Processing, Volume 1, Pages 900–903 (2002). https://doi.org/10.1109/ICIP.2002.1038171
5. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., Dally, W. J.: EIE: efficient inference engine on compressed deep neural network. In ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Volume 44, Issue 3, Pages 243-254 (2016). https://doi.org/10.1109/ISCA.2016.30
6. Felzenszwalb, P. F., Girshick, R. B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. In IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 32, Issue 9, pages 1627-1645 (2010). https://doi.org/10.1109/TPAMI.2009.167
7. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In proc. of the IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587 (2014). https://doi.org/10.1109/CVPR.2014.81
8. Girshick, R.: Fast r-cnn. In proc. of the ICCV conference, pages 1440–1448 (2015). https://doi.org/10.1109/ICCV.2015.169
9. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards realtime object detection with region proposal networks. In NIPS, Volume 39, Issue 6, Pages 91–99 (2015). https://doi.org/10.1109/TPAMI.2016.2577031
10. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Pages 779–788 (2016). https://doi.org/10.1109/CVPR.2016.91
11. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017). https://doi.org/10.1109/CVPR.2017.690
12. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
13. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A. C.: SSD: Single shot multibox detector. In European Conference on Computer Vision (ECCV), Pages 21–37 (2016). https://doi.org/10.1007/978-3-319-46448-0 2
14. Krizhevsky, A., Sutskever, I., Hinton, G. E.: ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems, Volume 1, Pages 1097-1105 (2012). https://doi.org/10.1145/3065386
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for largescale image recognition. arXiv Preprint, arXiv:1409.1556 (2014)

16. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Pages 1–9 (2015). https://doi.org/10.1109/CVPR.2015.7298594
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Pages 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
18. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision (IJCV), Volume 115, Issue 3, Pages 211-252 (2015). https://doi.org/10.1007/s11263-015-0816-y
19. Young, T., Hazarika, D., Poria, S., Cambria, E.: Recent Trends in Deep Learning Based Natural Language Processing. IEEE Computational Intelligence Magazine, Volume 13, Issue 3, Pages 55-75 (2018). https://doi.org/10.1109/MCI.2018.2840738
20. Han, J., Zhang, D., Cheng, G., Liu, N., Xu, D.: Advanced Deep-Learning Techniques for Salient and Category-Specific Object Detection: A Survey. IEEE Signal Processing Magazine, Volume 35, Issue 1, Pages 84-100 (2018). https://doi.org/10.1109/MSP.2017.2749125
21. Rosebrock, A.: Deep Learning for Computer Vision with Python. 1st edn. Published by Pyimagesearch (2017)
22. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Pages 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848
23. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 (2017).
24. Mittal, S.: Power management techniques for data centers: a survey. arXiv:1404.6681 (2014)
25. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L. C.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. arXiv:1801.04381 (2018)
26. Ghazi, P., Happonen, A. P., Boutellier, J., Huttunen, H.: Embedded Implementation of a Deep Learning Smile Detector. 7th European Workshop on Visual Information Processing (EUVIP)(2018). https://doi.org/10.1109/EUVIP.2018.8611783
27. Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size. arXiv:1602.07360 (2016)
28. Han, S., Mao, H., Dally, W. J.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149v3 (2015)
29. Wu, B., Wan, A., Iandola, F., Jin, P. H., Keutzer, K.: SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. arXiv:1612.01051v4 (2019)
30. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, Pages 6848–6856 (2018). https://doi.org/10.1109/CVPR.2018.00716
31. Wang, R. J., Li, X., Ling, C. X.: Pelee: A Real-Time Object Detection System on Mobile Devices. arXiv:1804.06882, Pages 1963–1972 (2018)