University of Padova
Department of Information Engineering

Master's Degree Thesis in

COMPUTER ENGINEERING

Deep Learning Networks for Real-time Object Detection on Mobile Devices

Author:
Andrea Gaetano Tramontano

Supervisor:
Prof. Stefano Ghidoni

Co-Supervisor:
Prof. Nicola Bellotto,
Ing. Jaycee Lock,
University of Lincoln, UK

Academic year 2018/2019

Acknowledgements

I would like to thank: my supervisor Stefano Ghidoni that follow me during all the thesis project with sympathy; professor Nicola Bellotto, Ing. Jaycee Lock, and all the friends from LCAS lab that host me warmly in the University of Licoln UK; all my friends from Italy, that did not believe that I would ever graduate; and finally my girlfriend and my family for supporting me and to bear me.

Abstract

Deep Neural Networks begin to be used in the last ten years in computer vision, and now they are the state-of-the-art for object detection. The majority of the researchers tackling the object detection problem have worked to improve the speed and the accuracy of DNNs, at the expense of memory consumption. However, a number of fields requires light networks capable to run on mobile and wearable devices. The Activis project, led by Professor Nicola Bellotto and Ing. Jaycee Lock from University of Lincoln, UK, works to provide a practical help for visual impaired people, developing an Android application to navigate in indoor environments. The need for an Object Detection model for this purpose is clear. In this thesis work are proposed and analyzed the performances of two models: SSD-Lite with Mobilenet V2 and Tiny-DSOD. They are two lightweight DNNs that promise to work in real-time and with a good accuracy on mobile devices. It is also proposed a new test dataset, called Office dataset, useful to test the two models in real-life indoor environments and as support for related researches. The final solution is currently used in the Activis project as object detector. The collaboration with the University of Lincoln UK, has brought to the publication of the paper about Activis project (and the proposed object detector) to the ICIAP 2019 conference.

Contents

Chapter 1

Introduction

The increasing performance of object detection systems brings this topic to be studied by more and more researchers in the last years [48] [84]. Nowadays it is possible to use object detection for complex applications and always more frequently new scenarios begin to need these features. Thanks to this, an increasing number of systems exploits object detection to improve the efficiency and to introduce automatism. As a fundamental part in computer vision and visual understanding, object detection is among the key components for solving more complex or high level vision tasks such as: segmentation, scene understanding, object tracking (Fig. 1.1a), image captioning, event detection, and activity recognition. Object detection has a wide range of applications in many areas of artificial intelligence and information technologies, including robot vision, consumer electronics, security, autonomous driving (Fig. 1.1b), human computer interactions, content based image retrieval, intelligent video surveillance (Fig. 1.1c), augmented reality, agriculture robotics, and medical analysis. Here are reported some more examples of uses to understand the spread of object detection around different environments. Face Detection (Fig. 1.1d), one of the most frequent, it is a technology used from social networks to tag people into pictures, but it can be found in smartphones to recognize the owner and unblock the device. Video surveillance system (Fig. 1.1c) aims to recognize people, animals, and various objects and it sends an alarm, if needed. Pedestrian detection, used in the increasing field of self-driving car (Fig. 1.1a), is fundamental to recognize pedestrian and to avoid accidents. Tracking object (Fig. 1.1b) is adopted in the industrial environment by various type of robots, in order to follow moving objects and track people in videos. Anomaly detection has the purpose of understanding rare or anomalous events, like a falling person in a home environment. People Counting enumerates people in a plaza or in a stadium and it generates statistics or intervenes in dangerous situations.

A new unexplored field is the possibility to use object detection for disabled people. Here, it may have a wide range of uses. A point of interest might be helping disabled people with their mobility. A focus about this task will be found at the end of this chapter.

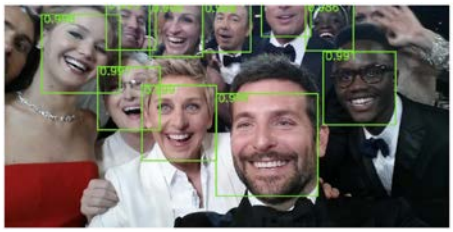
1.1 Object Detection Summary

Object detection can be grouped into two types [22], [83]: detection of specific instances and detection of specific categories. The first type aims to detect instances of a particular object (such as Zach Galifianakis's face, with and without beard, Fig. 1.2a), whereas the goal of the second type is to detect different instances of predefined object categories (for example humans, cars, bicycles, and dogs) (Fig. 1.2b). Historically, most of the researchers in the field of object detection has focused on the detection of a single category (such as faces and

(a)

(b)

(c)



(d)

Figure 1.1: Examples object detection uses: (a) self-driving car [77] (b) object tracking [5] (c) video surveillance [78] (d) face detection [34].

pedestrians) or a few specific categories. Eventually, in the last years the research community has started moving towards the challenging goal of building general purpose object detection systems with the purpose to reach a performance similar to the human recognition. However in 2012, Krizhevsky et al. [36] proposed a Deep Convolutional Neural Network (DCNN) called AlexNet which achieved record for image classification accuracy in the Large Scale Visual Recognition Challenge (ILSRVC) [64]. Since that time the research focus in many computer vision application areas has been on deep learning methods. Many new approaches based on deep learning born for generic object detection [19], [25], [18], [67], [63] and big progress has been achieved.

The second distinction in the object detection field is between General object detection and Salient object detection (Fig. 1.3). Visual saliency detection, aims to highlight the most dominant object regions in an image as illustrated in Fig. 1.3c. Numerous applications incorporate the visual saliency to improve their performance, such as image cropping [41] and segmentation [42], image retrieval [13] and object detection [20]. This type of object detection will not be explored, as it is not useful for our purpose. Indeed, we are not trying to highlight the main elements of a picture or to look at the most visible objects in it. The project purpose is to find one specific object, that could be not salient in the image. The generic object detection (Fig. 1.3b) problem itself is defined as follows: given an arbitrary image, determine whether there are any instances of semantic objects from predefined categories and, if present, to return the spatial location and extent. Object refers to a material thing that can be seen and touched. Generic object detection,

(a) (b)

Figure 1.2: Different object detection types: (a) instances object detection [70] and (b) category object detection [35].

differently from Object Detection in general, focus on approaches aimed at detecting a broad range of natural categories, as opposed to object instances or specialized categories (e.g., faces, pedestrians, or cars). Generic object detection has received significant attention, as demonstrated by recent progress on object detection competitions such as the PASCAL VOC detection challenge from 2006 to 2012 [11], [10], the ILSVRC large scale detection challenge since 2013 [64], and the MS COCO large scale detection challenge since 2015 [47]. The striking improvement in recent years is illustrated in Fig. 1.8.

1.1.1 Generic Object Detection

As said before, the generic object detection problem definition is related to determine where objects are located in a given image (object localization) and which category each object belongs to (object classification). The idea of locating an object results in the drawing of a bounding box around the object, i.e., an axis-aligned rectangle tightly bounding the object [11], [64]. Other possibilities are: a precise pixel-wise segmentation mask, or a closed boundary [65], [47], as illustrated in Fig. 1.4. To our best knowledge, in the current literature, bounding boxes are more widely used for evaluating generic object detection algorithms. While the classification results in a confidence value in percentage, that represents the similarity of an object to its category. We can divide the task of object detection in three stages: informative region selection, feature extraction and classification. These three stage are not always distinguished in a object detection system, but it is important to analyze their function to completely understand how the algorithm works.

1. **The Informative region selection**, have the role to define what in the image is significant and what not, in what area could be present an object and in which there is only the background (or irrelevant objects). Because many objects can appear in different positions, with different aspect ratio and different size in the image, it is necessary to scan the whole image to find possible objects. The scanning is made by windows of different size and aspect ratio (trying to detect different size and shape of objects). This strategy is called multiscale sliding windows and it is the most exhaustive one. Indeed, it can find out all possible positions of the objects. Due to the large number of generated windows, this

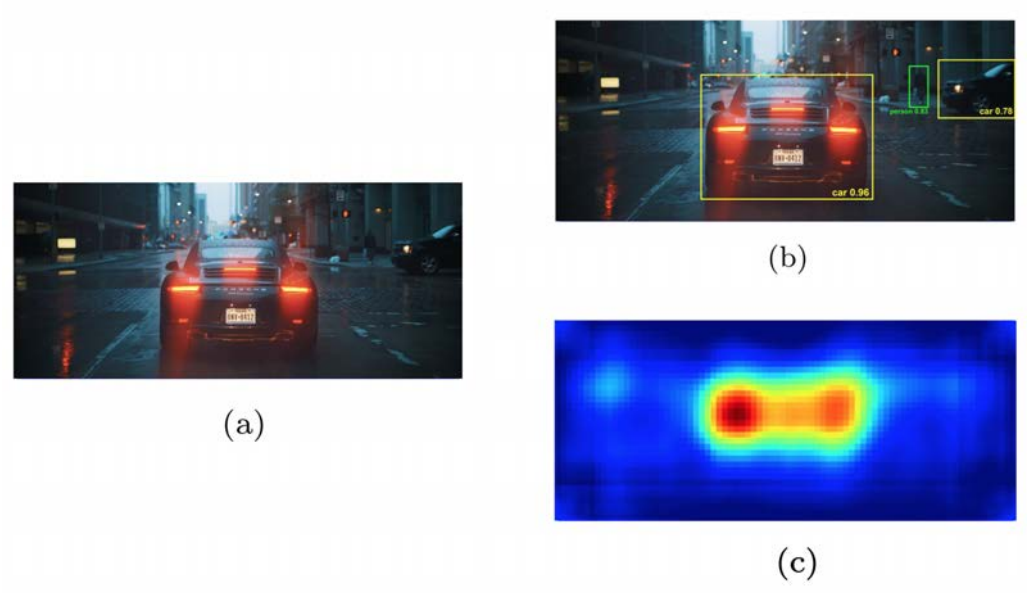


Figure 1.3: (a) An image processed with algorithm of (b) generic object detection and (c) Salient object detection. [37]

technique is computationally expensive and produces many redundant results. Viceversa, using only a little number of windows, the algorithm misses many important regions.

2. **Feature extraction**: to recognize (classify) object category, the algorithms extract features from images, that provide a robust representation of the objects. Each object could be defined by its own set of features. SIFT [52], HOG [7], and Haar-like [45] features are the famous computer vision algorithms that find in the objects the main features. With the upcoming of Deep Learning they were substituted, but they are still used in some fields like instances object detection. Since the different appearances, illumination condition and backgrounds of objects, it is very difficult to create and use a robust and universal features descriptor that match perfectly all types of objects.

3. **Classification**: the classifier is needed to differentiate a target object by all the other ones and to give an informative representation to the class category. Usually, used algorithms are: Support Vector Machine (SVM) [4], AdaBoost [16] and Deformable Part-based Model (DPM) [14].

This architectures obtained state-of-the-art results in the PASCAL VOC object detection competition [11], reaching real-time performance and a quite good hardware efficiency. However, the performance fastly increases using ensemble systems (putting together the three stages) and using little variants of this famous methods [19]. This fact is due to two main reasons:

1. Sliding windows strategy is redundant, inefficient and inaccurate.

2. There is a big drawback in creating manually the low-level descriptors and train the system with that parameters.

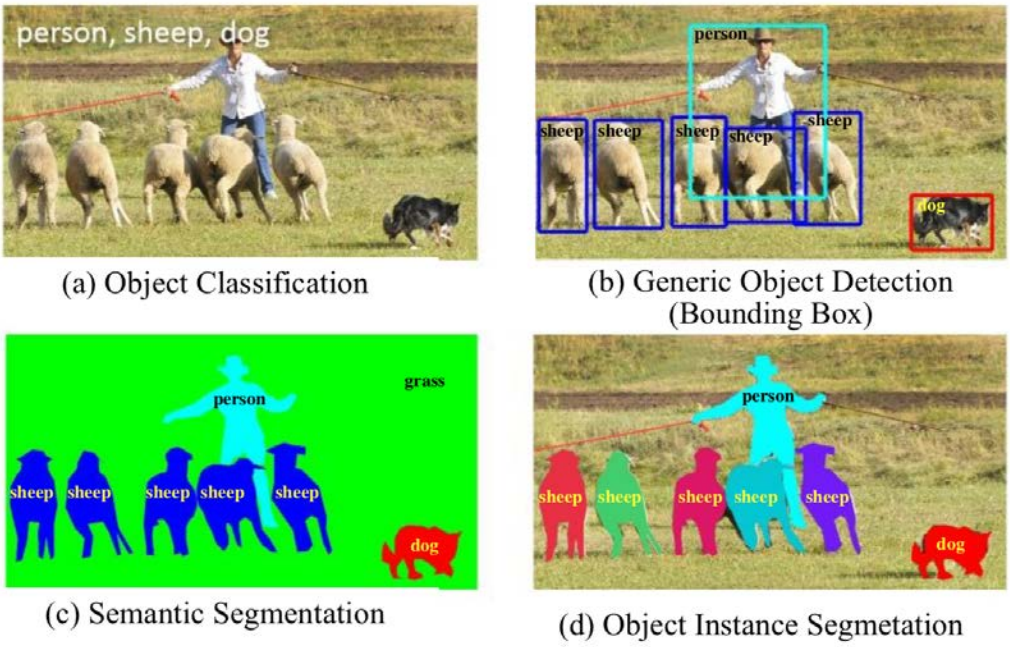


Figure 1.4: Recognition problems related to generic object detection. (a) Image level object classification, (b) bounding box level generic object detection, (c) pixel-wise semantic segmentation, (d) instance level semantic segmentation. [48]

An even more significant improvement is achieved thanks to Deep Neural Network (DNNs), precisely with the introduction of the Regions concept and with the CNN features (R-CNN) [41]. Deep learning begins to become famous fundamentally thanks to:

1. the emergence of the Imagenet [8] dataset, that with its large scale annotated training data, exhibits its big learning capacity.

2. the increasing performance in parallel computing system, like the GPU clusters.

3. the design of new network structures and new training strategies, found the base for famous algorithms such as: AlexNet [36], Overfeat [67], GoogLeNet [71], VGG [69] and ResNet [26].

4. The CNN are different from the traditional approaches and are able to learn a more complex features, thanks to the deeper architecture of those systems.

5. Also, thanks to the more robust training algorithms, CNN have a large expressivity and it is not necessary to design manually the features.

The ideal goal of generic object detection algorithm is to achieve high accuracy and high efficiency (Fig. 1.5). As shown in Fig. 1.5, a detector with high accuracy it is able to localize and recognize object in images: the object is correctly distinguished from an object of different category (high distinctiveness) but it have to result very similar to an object of the same class (high robustness). High efficiency is related to the performance of the algorithm in term of reaching the highest speed (realtime performance) and using a low amount of memory and storage. It is possible to

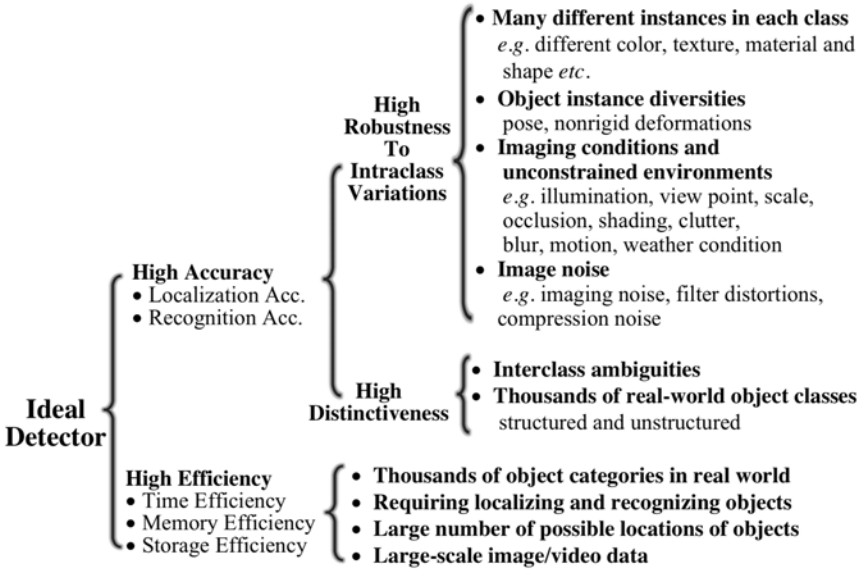


Figure 1.5: Summary of challenges in generic object detection [48].

see from the study of many algorithms [30] that, accuracy and efficiency is a trade-off not yet resolved.

Accuracy challenges

As mentioned before, the Accuracy challenges is made by:

1. the big range of intraclass variation;
2. the big amount of object categories.

The intraclass variation is divided in two types: intrinsic factors and image conditions. The first type is related to the fact that an object category can have elements with many type of different shapes, colors, sizes, materials and more. This is the example of the Chair category (Fig. 1.6h), in which very different types exist. Also humans could be difficult to analyze due to the fact that they dress differently. Instead, the second type depends on different image conditions and unconstrained environments, which can drastically change the appearance of the object. Many could be the factor, such as different times, locations, weather conditions, cameras, backgrounds, illuminations, viewpoints, and viewing distances. All these factors can produce variations in the appearance of the object: illumination, pose, scale, occlusion, background clutter, shading, blur and motion (examples illustrated in Fig. 1.6a-g). More challenges may be added by digitization artifacts, noise corruption, poor resolution, and filtering distortions. Finally, the huge amount of object category (10^4 - 10^5) requests an high discriminating power to the detector, to see difference between classes of object that are very similar (1.6i)). That been said, current detectors focus mainly on structured object categories, such as the 20, 200 and 91 object classes in PASCAL VOC [11], ILSVRC [64] and MS COCO [47], respectively. Clearly, the number of object categories under consideration in existing benchmark datasets is much smaller than that can be recognized by humans.

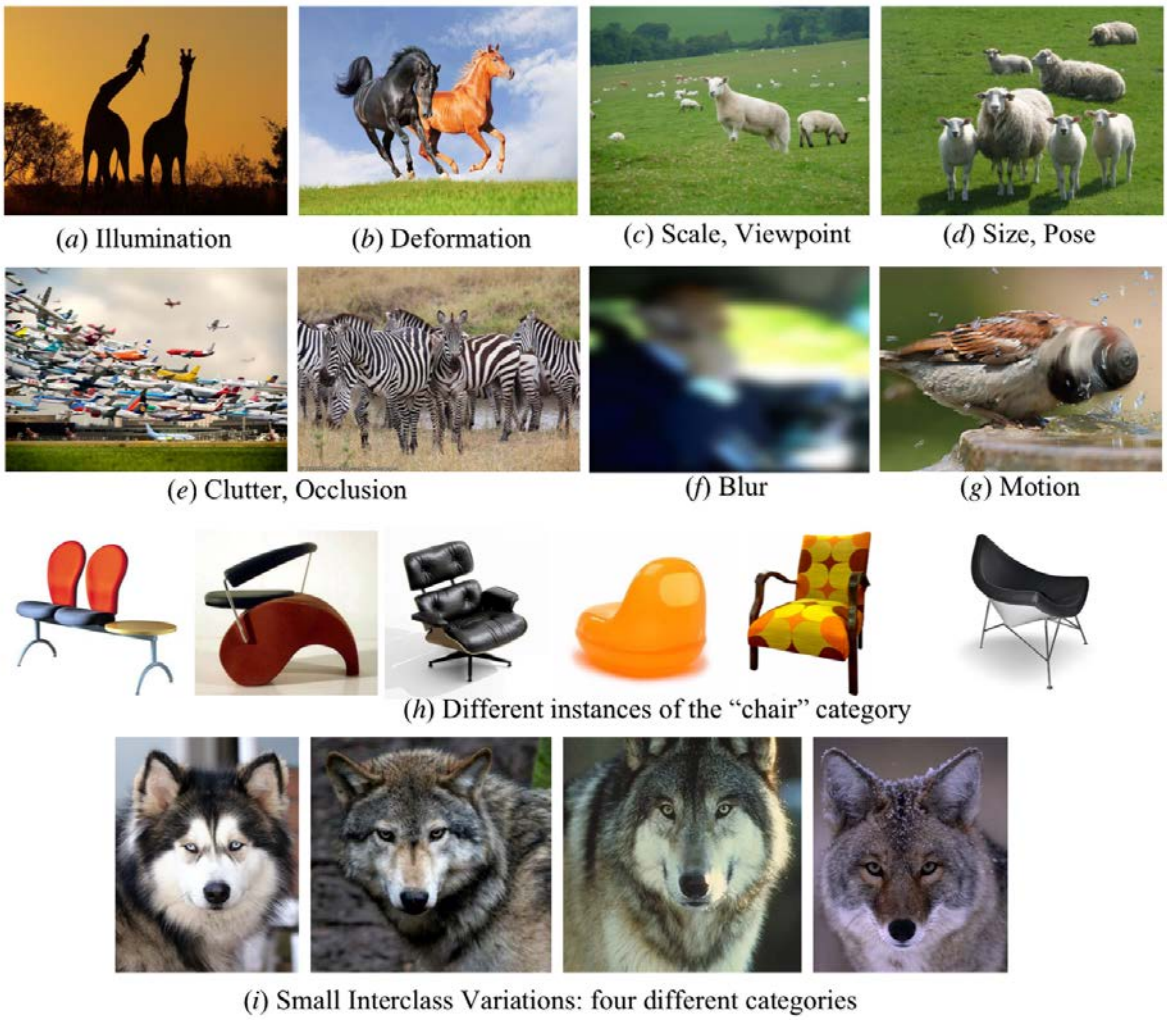


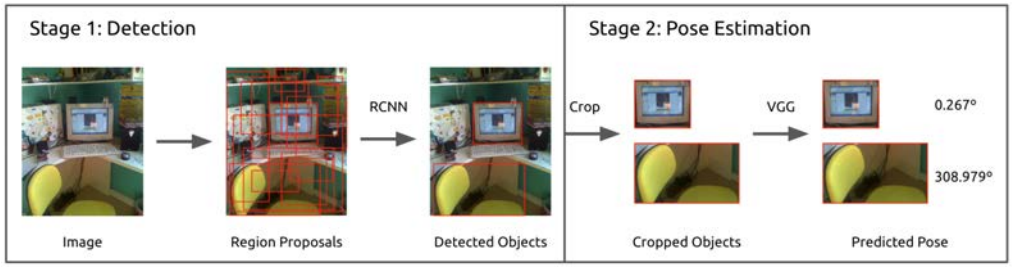
Figure 1.6: Changes in imaged appearance of the same class with variations in imaging conditions (a-g). There is an astonishing variation in what is meant to be a single object class (h). In contrast, the four images in (i) appear very similar, but instead are from four different object classes. Images from ImageNet [64] and MS COCO [47]. [48]

Efficiency challenges

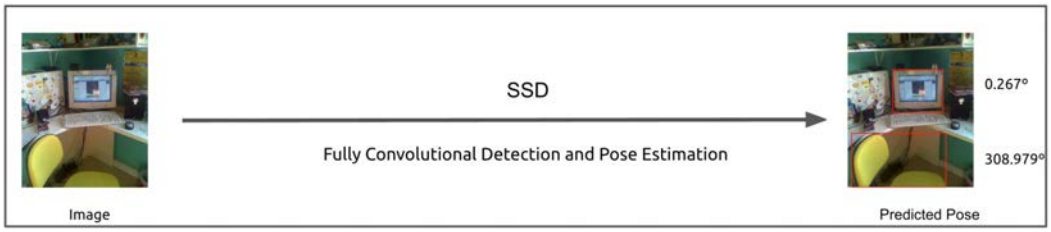
The exponentially increasing number of images arriving from the social and mobile world ask for a efficient and scalable detector. The majority of images come from smartphone and social media networks and both have limited computational power. In that cases an efficient detector is crucial. Here the challenge is to locate and recognize objects that are present in a single image at different scale. Moreover, the scalability is another challenge, because the increasing number of different objects ask to a detector to be able to handle unseen objects, unknown situations and rapidly increasing image data. Thinking about the ILSVRC [64] database, there is a limit

to the possible manual annotations that are feasible to obtain. So, many times it is an algorithm that annotate the images and it can produce weaknesses and mistakes.

Generic object detection framework with Deep Learning networks, could be divided into two main categories: single-stage based methods (Fig. 1.7a) and two-stages based methods (Fig. 1.7b).



(a)



(b)

Figure 1.7: Two-stage vs. Proposed. (a) The two-stage approach separates the detection and pose estimation steps. After object detection, the detected objects are cropped and then processed by a separate network for pose estimation. This requires resampling the image at least three times: once for region proposals, once for detection, and once for pose estimation. (b) The proposed method, in contrast, requires no resampling of the image and instead relies on convolutions for detecting the object and its pose in a single forward pass. This offers a large speed up because the image is not resampled, and computation for detection and pose estimation is shared. [58]

The two-stages based methods use a "region proposal" approach: an external algorithm proposes the region of interest, that are later classified to recognize objects. They can reach better accuracy in respect to the other category, but they request more memory and more computational time. Related to this category are: R-CNN [19], SPP-net [25], Fast R-CNN [18], Faster R-CNN [63], R-FCN [6], FPN [46] and Mask R-CNN [24], some of which are correlated with each other (e.g. SPP-net modifies RCNN with a SPP layers).

The single-stage based methods instead apply different size of windows to the input image, so they are able to check if the object appears in the windows. These methods yield better trade-off between accuracy and speed. Usually they are faster and less accurate then the two-stage methods. Sometimes they also use a backbone network as features extractor, that helps to reach better accuracy. On the other side it increases the parameters of the network and cause greater computational costs. This method mainly includes: MultiBox [9], YOLO [59], SSD [49], YOLOv2 [60], YOLOv3 [62], and DSOD [68]. The correlations between these two pipelines are bridged by the anchors introduced in Faster RCNN.

The best object detectors nowadays are based on Deep Learning technologies to achieves the goal. Deep Convolutional Neural Networks (DCNN) have reached good results in the last years. The general trend brings to the creation of complicated and heavy networks to increase

the detection accuracy, such as Faster Region Convolutional Neural Network (Faster R-CNN) [63], You Only Look Once (YOLO) [59] [60] [62], Single Shot Detection (SSD) [49] and all their variants. Most of these networks recognize very well objects in images and videos (with good accuracy), but they are high complexity models. These systems require a big amount of operations to predict a result, which yields an increase of power consumption. Moreover, a complex network brings to the use of many resources (in terms of computation and memory) or to an high computational time.

As confirmed in section 4, these networks are not very useful for our goal, because this work aims to use a smartphone as support (low performance device). To resolve the issue, networks as Mobilenet come in aid. It first introduces the use of Depth-wise Separable Convolution, that decreases the amount of operations for a single convolution and the complexity of the network itself. Therefore, the network decreases its size and requires less memory, with a positive impact on computational time, needed resources and energy-consumption. On the other side, the accuracy decreases and this is the big cons of these systems: it is not possible to have high accuracy with high speed and low amount of resources. The trade-off between accuracy and speed have been deeply studied in [30].

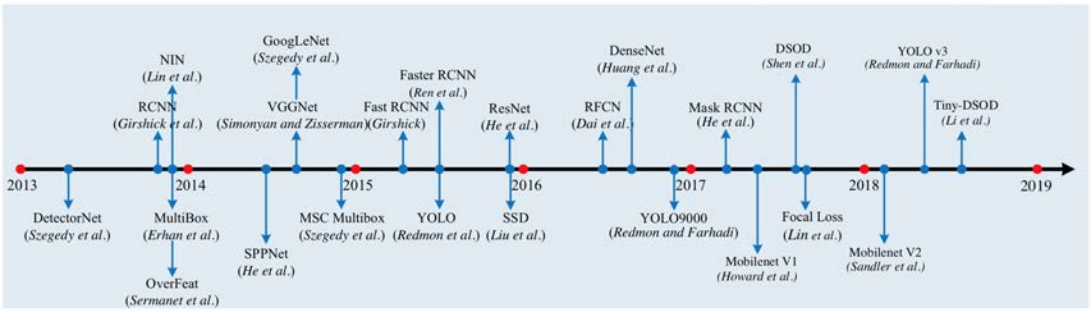


Figure 1.8: Milestones of object detection and recognition.

Exist common measures to describe an object detection models. It is important to use the same type of measures and with the same unit, to help the comparison between different models. Typical used measures and also used in this thesis work are:

- **mAP**: the mean Average Precision is the most used measure to compute the accuracy of a model. It is the mean of the Average Precision of each object categories. The Average Precision is computed for each object using the precision and the recall of the same. Because of that it is a good representation of the accuracy.
- FLOPS: the FLoating point Operations Per Second is the number of floating-point operations that an algorithm have to perform to terminate. This measures is more accurate then the number of instructions per second.
- **Params**: the parameters are the coefficients of a network and they are chosen by the network itself, during the training process. The number of parameters (Params) define the complexity of the network: more parameters mean more complexity and more possibility to learn better, but also more need of images example for training and more heavy network.
- **MAdds**: Multi-Adds id the number of multiply-accumulates needed to compute an inference on a single image and it is another common metric to measure the efficiency of the model.

- **FPS**: the Frame Per Second is the measure of time most used in object detection. It represent the number of frame that the network is able to process in a second.
- **CPU time**: the CPU time is an alternative measure of time, but it focus on the time needed to process a single frame.

1.2 The Activis Project

Inspired by the proposal Active Vision with Human in the Loop for People with Vision Impairment [32] [51] [50], in collaboration with Google, Professor Nicola Bellotto and Doctor Jaycee Lock form University of Lincoln, set a project to help blind people mobility in indoor environments. The goal of the task is to build an Android application easy to use, aiding to find objects of interest and navigate in a before unseen indoor setting (Fig. 1.9). The smartphone provided for the project is an Asus ZenFone AR with project TANGO. This smartphone is equipped with a depth sensor and a 3D motion tracking sensor (Fig. 1.9). An issue for blind or visual impaired

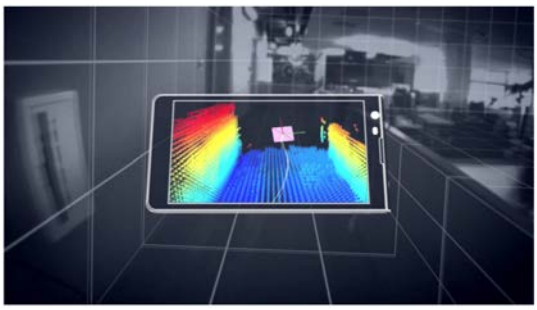


Figure 1.9: Idea of smartphone analyzing the indoor environment. [32]

people is to move in an indoor environment that they have never seen before, to navigate avoiding possible obstacles and to find objects around. The system proposed provides valuable help. Consider a blind person looking for a chair in a new office or in a generic room (Fig. 1.10a and 1.10b). It is next to the door and he wants to know where the chair is located. The phone, through the camera, can help in the localization of the chair and can drive the person to reach it. This is particularly challenging due to the unpredictability of human motion and sensor uncertainty. Because of that, to help the object detection, a statistics about objects location and the relationships between them is generated. So, if the detector is not able to detect a specific object, but it finds objects that are known to be close to the target one, it can advice the user to move near them to better detect the target object from the new position. It is clear the importance of using smartphones as support, instead of more complex or bigger devices. One purpose of the project is to provide a user-friendly interface, as easy as possible to interact with. This project could help blind people in a independent mobility in indoor location and could decrease the time for many operations.

1.3 Thesis Goals

Followed by Professor Stefano Ghidoni and in collaboration with Professor Nicola Bellotto and Doctor Jaycee Lock from University of Lincoln, UK (LCAS lab), for the development of the Activis project [33], this thesis work provides a possible solution to identify objects in video

Figure 1.10: The system in use during an experiment with a blindfolded participant [33].

frames from a low performance device. In this type of applications object detection has a primary importance for the recognition and localization of item and possible obstacles in the room. The features we are looking for our algorithm are:

- it have to work on an Android smartphone;
- it have to be able to recognize a group of twenty different objects;
- it have to be robust to anomalous movements of the user;
- it have to avoid as many errors of detection as possible.

We decide to focus on office room environments (Fig. 1.10a and 1.10b), but it is easy to find possible uses in other type of room like public place, kitchen or bathroom and many other users unexplored environments. We choose to compare the state-of-the-art networks for low performance devices and some novel proposal of the last years. We are looking for the best features needed to reach a compromise between speed and accuracy. Furthermore, because of the novelty of this application field, the thesis work lays the foundation for future experiments, providing instruments to evaluate new proposal networks and compare them with the state-of-the-art. For these reasons we provide a new data-set of office images and three videos, to test the networks performances. Indeed, a test made only on images, does not take care about the noise introduced by the user movements, that can produce a blur on the video frames and create more tricky cases. This is why it is important to test the application on videos too: the aim of the thesis work is not only to find the best speed-accuracy trade-off, but also to find a robust system. The remainder of the thesis is organized as follows: section 2 presents the state-of-the-art for object detection and the related work made till now; section 3 describes the architecture of the three models and explains the main difference between them; section 4 analyzes the results of the test; section 5 will summarize the work and takes conclusions.

Chapter 2

Related Work

This section will give a panorama about the most famous algorithms in the object detection fields. Each of these algorithms are the best in its challenge type (accuracy, efficiency, speed) or they are famous algorithms that founded the base for the nowadays state-of-the-art. It is important to understand the mechanisms at the base of this networks and to discern its qualities and weakness, to make right choice in future novel proposals. The chapter will describe the two-stage based model, continue with the one-stage based ones and finally complete with some light networks, that better fit the purpose of the thesis.

2.1 Two-staged based models

The most famous two-staged based network is **RCNN** (Fig. 2.1). Inspired by the good results obtained with CNN networks, RCNN proposes a Region Proposal architecture for generic object detection. It integrates AlexNet [36] with the region proposal method Selective Search [76]. This method was the first to achieve high quality accuracy, but it has also notable drawback [25]:

1. Training is a multistage complex pipeline, which is slow and hard to optimize, because each individual stage must be trained separately.
2. Numerous region proposals which provide only rough localization need to be externally detected.
3. Training SVM classifiers and bounding box regression is expensive in both disk space and time, since CNN features are extracted independently from each region proposal in each image.
4. Testing is slow, since CNN features are extracted per object proposal in each testing image.

In the following years new networks were created based on RCNN. Each new proposal try to increase performances or accuracy of the precedent ones, finding solutions to the new generated bottlenecks. SPPNet [25] introduce the traditional spatial pyramid pooling (SPP) [21] [40], to significantly speed up without sacrificing any detection quality. Fast RCNN [18] enables end-to-end detector training by developing a streamlined training and introduce the idea of region proposals with the Region of Interest (RoI). The Faster RCNN framework (Fig. 2.22) proposed by Ren et al. [63] proposed an efficient and accurate Region Proposal Network (RPN) to generating region proposals. They utilize a single network to accomplish the task of RPN for region proposal and Fast RCNN for region classification. Faster-RCNN is able to reach a speed of 7 FPS with

an accuracy of 73,2 mAP in the COCO dataset challenge. The last ones are RFCN (Region based Fully Convolutional Network) [6], Mask RCNN [24] and Light Head RCNN [44], all based on Faster-RCNN. They increased the speed in respect to Faster-RCNN, bringing to a better trade-off between accuracy and performances.

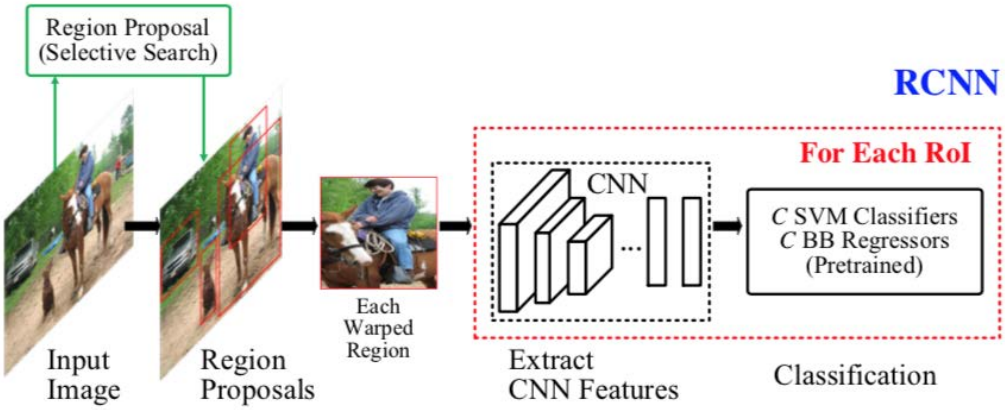


Figure 2.1: High level diagrams of the RCNN frameworks [48].

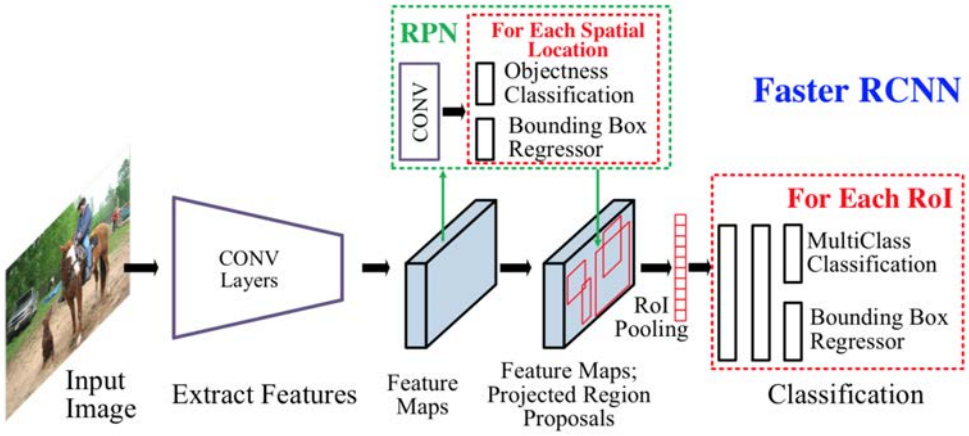


Figure 2.2: High level diagrams of the Faster-RCNN frameworks [48].

2.2 Single-staged based models

Region based approaches are computationally expensive for low performance devices, because they have limited memory, storage and computational power. Researchers have begun to develop unified detection strategies, trying to reach better efficiency. Unified structures relies to architecture that directly compute classification and detection from the images with a single forward on the CNN network. It does not involve region proposal systems or post classification layers. This

type of networks can be divided in two sub-networks, one dedicated to extract the features from the input and one to classify and locate the objects. The former part is called backbone network, while the latter one is usually called front-end network. This structure helps the generation of object detector networks based on pre-existing classification networks. Indeed in many case the development of the final architecture embed some layers of a classification model: usually are deleted the final layers producing the classification output and only the features extractor layers remain. Highlight the difference from the two-stage based methods, where classification and detection are computed by two different networks. Instead the single-stage based methods (with a backbone network), could use as base network a piece of a classification network (the features extractor one), but finally compute classification and detection at one time in the lasts layers. This is the case of all the following networks, that have their own backbone subnetwork.

YOLO (You Only Look Once), is nowaday the best object detection network, working at real-time performances. Redmon et al. [59] proposed YOLO as unified detector, producing a regression from image pixels to a spatially separated bounding boxes and an associated class probabilities. The design of YOLO is illustrated in Fig. 2.3. YOLO doesn't use a generative stage for region proposal, instead it directly predicts detections using a small set of candidate regions. Unlike region-based approaches, e.g. Faster RCNN, that predict detections based on features from local region, YOLO uses the features from entire image globally. In particular, YOLO divides an image into a S×S grid. Each grid predicts C class probabilities, B bounding box locations and confidences scores for those boxes. These predictions are encoded as an S×S×(5B + C) tensor. By throwing out the region proposal generation step entirely, YOLO is fast by design, running in real time at 45 FPS and a fast and light version, i.e. Tiny YOLO [59], running at 155 FPS and with a light architecture. Since YOLO sees the entire image when making predictions, it implicitly encodes contextual information about object classes and is less likely to predict false positives on background. YOLO makes more localization errors resulting from the coarse division of bounding box location, scale and aspect ratio. As discussed in [59], YOLO may fail to localize some objects, especially small ones, possibly because the grid division is quite coarse, and because by construction each grid cell can only contain one object. YOLOv2 and YOLO9000 are the following updates by Redmon and Farhadi [60]. They are an improved version of YOLO, in which the custom GoogLeNet [71] network is replaced with a simpler DarkNet19 (19 layers), plus utilizing a number of strategies drawn from existing work, such as batch normalization [78], removing the fully connected layers, and using good anchor boxes learned with k-means and multiscale training. YOLOv2 achieved state-of-the-art on standard detection tasks, like PASCAL VOC and MS COCO. In addition, Redmon and Farhadi [60] introduced YOLO9000, which can detect over 9000 object categories in real time by proposing a joint optimization method to train simultaneously on ImageNet and COCO with WordTree to combine data from multiple sources. Finally the last version YOLOv3 [62] (released after SSD publication) introduce the multiscale widows generation to produce more accurate result and increase its speed substituting the backbone network with the new Darknet-53 having 53 layers. The version 608 of YOLO reaches an accuracy of 60 mAP in the COCO challenge with a speed of 20 FPS. There exists more than one version of YOLO with different performances and accuracy values as illustrated in Table 2.1. As said before, Yolo is the state-of-the-art in real-time object detection, but the real-time performance are reached with high cost in terms of hardware, indeed all the test and results are obtained with a Nvidia Titan X GPU.

SSD (Single Shot Detector) proposed by Liu et al. [49] try to preserve real-time speed without sacrificing too much detection accuracy (Fig. 2.4). It is faster than YOLO v1 [174] and has accuracy competitive with state-of-the-art region-based detectors, including Faster RCNN [63]. SSD effectively combines ideas from RPN in Faster RCNN [63], YOLO [59] and multiscale CONV features [23] to achieve fast detection speed while still obtain high detection quality. Like

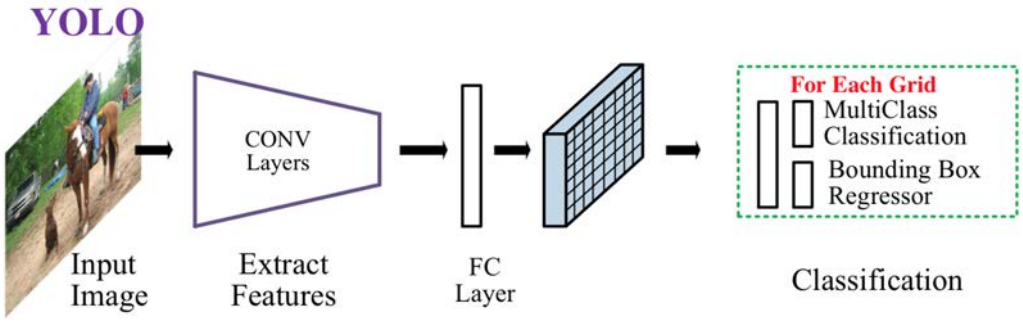


Figure 2.3: High level diagrams of the YOLO frameworks [48].

YOLO, SSD predicts a fixed number of bounding boxes and scores for the presence of object class instances in these boxes, followed by anNo Maxima Suppression (NMS) step to produce the final detection. The CNN network in SSD is fully convolutional, whose early layers are based on a standard architecture, such as VGG [69] (truncated before any classification layers), which is referred as the backbone network. Then several auxiliary CONV layers, progressively decreasing in size, are added to the end of the base network. The information in the last layer with low resolution may be too coarse spatially to allow precise localization. SSD uses shallower layers with higher resolution for detecting small objects. For objects of different sizes, SSD performs detection over multiple scales by operating on multiple CONV feature maps, each of which predicts category scores and box offsets for bounding boxes of appropriate sizes. For a 300 ÃŮ 300 input, SSD achieves 41.2 mAP on the COCO challenge at 46 FPS on a Nvidia Titan X.

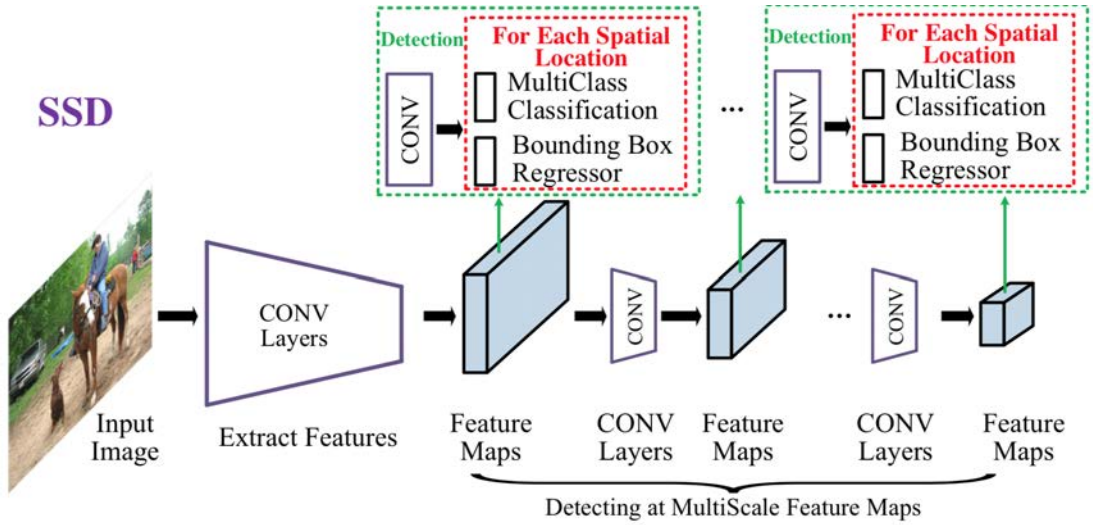


Figure 2.4: High level diagrams of the SSD frameworks [48].

2.3 Light networks

Mobilenet [28] is the first work to introduce Depth-wise Separable convolution, that consists in a Depthwise convolution followed by a Point-wise one (kernel 1×1) as illustrated in Fig. 2.5. Computing the convolution in two steps decrease the numbers of operations needed. Mobilenet is a classification network, but can be used as backbone for an object detection system. The light features of the Depthwise convolution yield to reduce drastically the memory usage and consequently the model obtains better speed performance with low resources devices. On the other side, we are losing accuracy, because Depth-wise Separable convolution create a gap in term of precision. In Tab. 2.1 we can see the performance of the described network in terms of accuracy, computational speed and model size, to have a view of what we have just reported. After Mobilenet, other researchers produced new models using Depth-wise Separable Convolution and reducing the complexity. One good example is the following network.

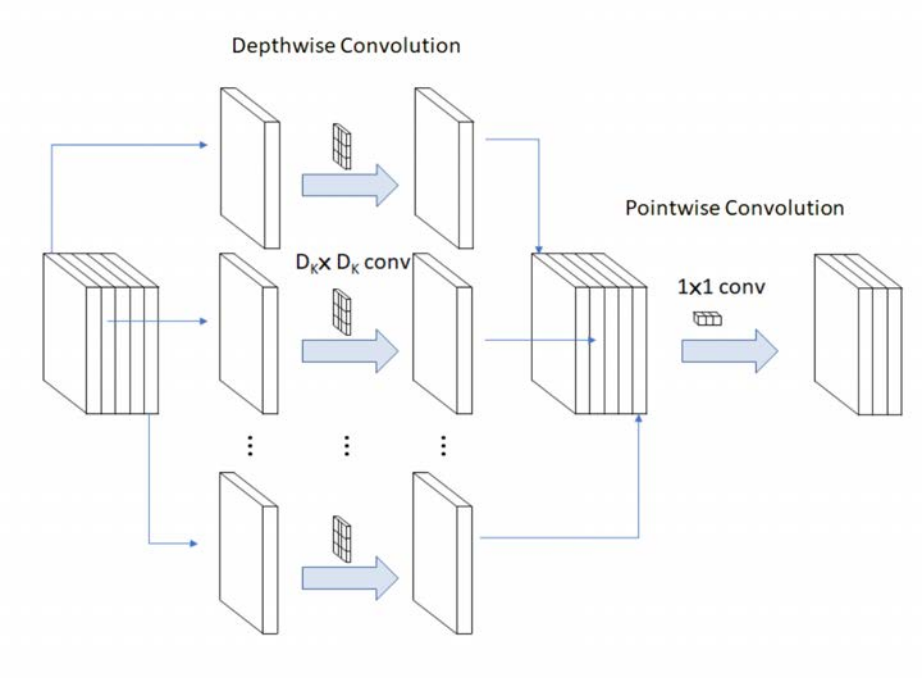


Figure 2.5: The Depthwise Separable convolution, composed by a Depthwise convolution and a Pointwise one [73].

Tiny-DSOD [43] try to obtaining a good balance between the used resources (FLOPs and memory) and accuracy trade-off. The backbone part of the proposed framework is inspired by the object detection work DSOD [68] and the depthwise separable convolution network structures. DSOD [68] introduces several important principles to train object detection network from scratch and adopt a deep supervision like structure from DenseNet [29]. Tiny-DSOD combines the depthwise separable convolutions into the DenseNet structure, and introduces a novel depthwise dense block (DDB) to replace dense blocks in DenseNet. This design promise to reduce the computing resource requirements, but also bring to an efficient training. For the front-end part, Tiny-DSOD use the feature-pyramid-network (FPN) [46] to produce semantic information from low-resolution scale and from high resolution scale, combining later them to generate a more

precise result. Again they use Depthwise convolution in the FPN structures to increase the speed performance of the network and to decrease the use on memory (see Tab. 2.1).

Some others novel proposals are: LightFlow [85] that compare different frames in the video to detect object only from the best pictures and Effnet [15] that based on Mobilenet introduces a new Convolutional block to increase the computational speed and decrease the complexity of the system.

Method	**COCO mAp (0.5:0.95 IoU)**	**FPS (Titan X)**	**Params**	**FLOPS**
Faster-RCNN	21.5	7	134.70M	181.12Bn
YOLOv3 - 320	51.5	45	-	38.95Bn
YOLOv3 - 608	57.9	20	-	140.69Bn
Tiny-YOLOv3	33.1	220	-	5.56Bn
SSD300	25.1	46	36.1M	31Bn
SSD512	28.8	19	36.1M	91Bn
SSDLite (Mobilenet V1)	22.2	>1000	1.3M	5.1Bn
SSDLite (Mobilenet V2)	22.1	>1000	4.3M	0.8Bn
Tiny-DSOD	23.2	>1000	1.15M	1.12Bn

Table 2.1: Comparison between state-of-the-art networks. TODO: add missing network (tiny-yolo, other yolo version).

Chapter 3

Architecture

Before discuss about the networks chosen for the thesis comparison, it is important to highlights because some models have been discarded from this analysis and instead because others ones have been selected.

3.1 Discarded models

The networks discarded from this analysis are:

- **Faster-RCNN** [63] (and all the networks based on it [24] [44] [6]): it is a two-stage based network, so as seen in the previous chapter 2, it is not able to reach real-time performances with a Nvidia Titan X GPU. Consequently, it is not possible that it reaches real-time performances in a mobile platform. The weight of this model, in terms of FLOPs, is of $181.12Bn$ and its speed is of $7FPS$ (Tab. 2.1). Furthermore, networks like YOLOv3 are able to reach better performances in terms of speed, accuracy and weight.

- **YOLOv3** [62]: it is the state-of-the-art in real-time objects detection. It reaches the best accuracy of $60.6mAP$, with a good speed ($20FPS$) (Tab. 2.1). Again the drawback of this network is the weight. The YOLOv3-608 version have a weight, in term of FLOPs, of $140.69Bn$. So, it is not possible to run this model in a mobile device and obtain real-time performances.

- **Tiny-YOLOv3** [43]: YOLOv3 has also a light version. It reaches a speed of $220FPS$ (on a Nvidia Titan X), with a big drawback in terms of accuracy ($33.1mAP$). Its weight is really lower compared with its standard version, indeed it amount to $5.56Bn$, thirty times less than YOLOv3-608. Because of its low weight we decide to test Tiny-YOLO on the smartphone provided for the thesis experiments. We would like to understand, what speed the model was able to reach. We create a Tiny-YOLO network using the instruments provided by OpenCV version 4 [3] and by YOLO framework [61]. YOLO provide two file a .cfg file containing the configurations of the network and a .weights file containing the weights of the network trained on COCO dataset. Using OpenCV (with C++ language), it is possible to generate the network of Tiny-YOLOv3 from the .cfg file and put the weights in the correspondent layers through the .weights file. Then using the NDK tool [1] provided by Android, it is possible to integrate the C++ code in to the Android application. We create a simple Android script to test the network on the smartphone. The results were not thrilling: it finally reaches a speed of $2FPS$. The test on videos was not good, because the

network was able to recognize many object, but only if the smartphone stayed in a fixed position. Otherwise, also slow movements (and the consequently blur) create problem for the object recognition in the frames. This experiment demonstrate that the structure of this network is not able to reach real-time performances on mobile device.

- **LightFlow** [85]: it is a video object detector light network for mobile device. It proposes some interesting features and tries to resolve some common issue of the video detection. Its main structure is made by three components:
  - a light weight image object detector for sparse key frames;
  - a very small network, Light Flow, designed for establishing correspondence across frames;
  - a flow-guided GRU module designed to effectively aggregate features on key frames (for non-key frames, sparse feature propagation is performed).

  It is a powerful network promising to reduce the detection errors generated by blurred frames. This problem appears when the camera recording video is moving inconstantly (like a smartphone in a human hand). The only counterpart of this network is the training process. Indeed, it is needed a video dataset (like Imagenet vid [64]) to train the model. The purpose of the thesis is to use the application for an office environment and nowadays there not exist video datasets with this category of objects. This network is another possible choice for future comparison works, but without a right dataset as support, it is impossible to apply the model to our practical goal.

The two aforementioned networks are complete object detector systems, like Single Shot Detection [49]. SSD and YOLO are both single-stage based network, and as said in chapter 2 they both use a backbone network. YOLOv3 use the Darknet-53 [62] and SSD use VGG16 [69]. It is possible to change the backbone network of both YOLO and SSD. But it is known by experiments that, while SSD is more compatible with different types of backbones networks, YOLO is created to work well and close to its original backbone network and substituting the Darknet with another one, could hurt the performances in terms of accuracy and speed. In the following paragraph it will be presented some of the discarded classification networks, that could work well as SSD backbone networks.

- **ShuffleNet** [82] [53]: it is a light network introducing two new operations called Point-wise Group convolution and Channel Shuffle. It promise reduce computation costs while maintaining accuracy.
- **Effnet** [15]: it is a Mobilenet V1 based network. It proposes a new Depthwise Separable convolution Block to resolve weaknesses of Mobilenet and ShuffleNet. It reaches a better accuracy without increasing the network weight.

Both these models are possible solutions for the thesis project, but we decide to discard them because, as illustrated in Tab. 2.1, Tiny-DSOD [43] promises to perform better in terms of network weight and accuracy. They remain as possible candidates for a future comparison works.

3.2 Selected models

Finally, the choice for the object detection systems to compare and analyse, fall in these two networks:

- **SSD Lite with Mobilenet V2** [66]: Mobilenet V2 is the state-of-the-art network for real-time classification on mobile devices and it is well supported and studied from many researchers in the last years. SSD is one of the best front-end part for object detection and it is very adaptable with different backbone models. The features fusion of these two models bring to an efficient object detector, promising to work well and with a good accuracy.
- **Tiny-DSOD** [43]: it is the best novel proposal that promise to perform better then SSD Lite with Mobilenet V2 and then all the others networks in the environment.

3.3 Depthwise Separable convolution

There exist two types of Separable Convolution [79]:

- Spatial Separable Convolution;
- Depthwise Separable convolution.

But before describing these two convolutions, here an example of standard convolution to remember how it works.

3.3.1 Standard Convolution

Assume to have an input of $12 \times 12 \times 3$ pixels, an RGB image of size 12×12 (width, height and number of channels). Considering the width, height and depth of the image and applying a $5 \times 5 \times 3$ convolutional kernel with no padding and stride 1, the convolution produce a output of (12-5+1=8) $8 \times 8 \times 1$ pixels. Substantially, each time the convolutional kernel take $5 \times 5 \times 3$ pixels and produce one number from them. This means, that the total amount of multiplications are $5 \times 5 \times 3 = 75$ every time the kernel moves. If the desired result is to have an output of $8 \times 8 \times 256$ pixels (with 256 channels), it is possible to apply two-hundreds and fifty-six times the kernel of dimension $5 \times 5 \times 3$ and stack the results together (as illustrated in Fig. 3.1).

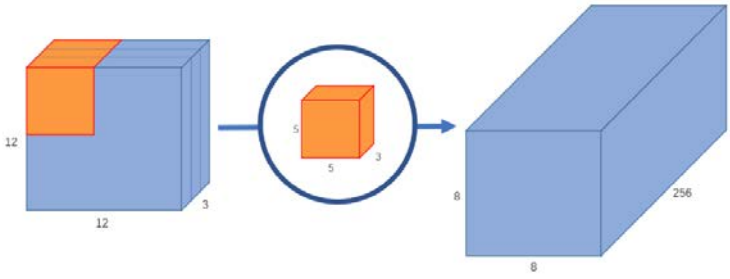


Figure 3.1: Standard convolution applying one 256 kernels [79].

3.3.2 Spatial Separable Convolution

The Spatial Separable Convolution divides a kernel in two sub-kernels, using only the width and the height dimensions (spatial ones) and not using the depth one. It is the case of the kernel illustrated in Fig. 3.2, divided in the two sub-kernels. In a standard convolution twenty-five operations are applied, while in the Spatial Separable one only five operations per kernel (ten

in total). This technique is not applicable to all the kernels, but only to the ones divisible into sub-kernels. Instead the Depthwise Separable convolution has the advantage to be applicable to all kernels and that why it is more used.

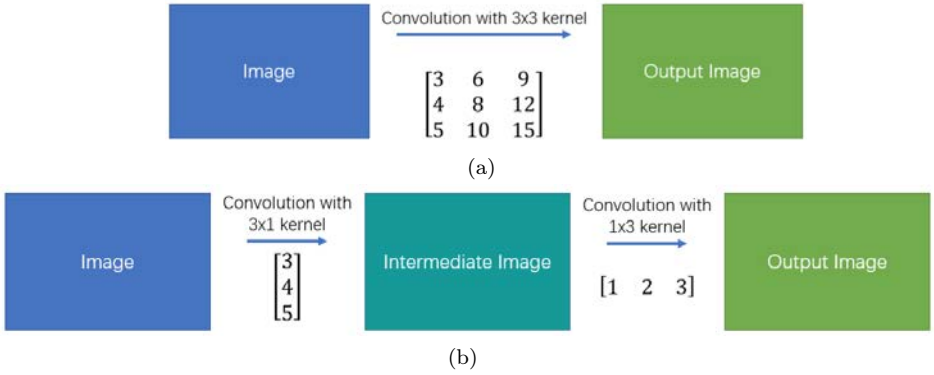


Figure 3.2: Standard convolution (a) and Spatial convolution with the correspondents sub-kernels (b) [79].

3.3.3 Depthwise Separable convolution

As illustrated in Fig. 3.3 and 3.4, the Depthwise Separable convolution is composed of operations (with two different kernels):

1. a Depthwise convolution;
2. a Pointwise Convolution (kernel 1×1).

Depthwise convolution consists in a convolution that doesn't change the depth dimension. So, using the previous example, it is possible to obtain this result applying three kernels of dimensions $5 \times 5 \times 1$ (Fig. 3.3). Each of the three kernels iterate on one channel of the input image (only one not all), getting the scalar product of every twenty-five pixel groups and producing a $8 \times 8 \times 1$ image. Stacking the three images (one per kernel) together generates a $8 \times 8 \times 3$ image.

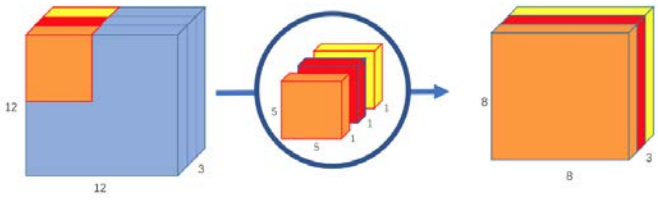
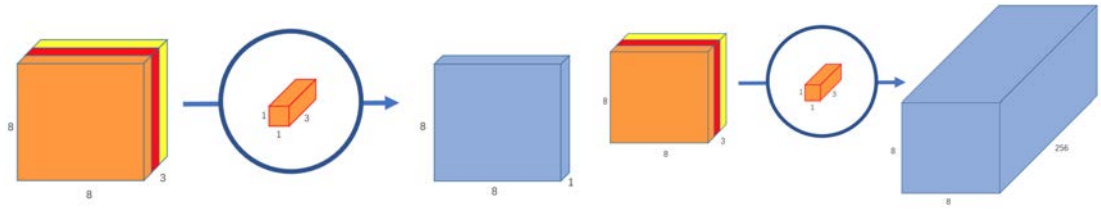


Figure 3.3: Depthwise convolution uses, 3 kernels to transform a $12 \times 12 \times 3$ image in a $8 \times 8 \times 3$ one [79].

Pointwise Convolution the original convolution transformed a $12 \times 12 \times 3$ image into a $8 \times 8 \times 256$ image. Currently, the Depthwise convolution has transformed the $12 \times 12 \times 3$ image to a $8 \times 8 \times 3$ image. Now, the number of channels have to increase in each image. The Pointwise Convolution uses a $1 \times 1 \times 3$ (for this example) kernel to iterate through every single point, for

the whole depth dimension. So, the kernel iterates on the $8 \times 8 \times 3$ image and produce an output of dimension $8 \times 8 \times 1$. Applying two-hundreds and fifty-six times the same kernel and stacking together the results, produce the final output of $8 \times 8 \times 256$.



(a) (a) (b) (b)

Figure 3.4: Pointwise convolution transform an image of 3 channels in an image of 1 channel (a) and repeating the same procedure with 256 kernels produce an image of size $8 \times 8 \times 256$ (b) [79].

It is easy to understand the advantages of Depthwise Separable convolution compared with the standard one calculating the number of operations for both the convolutions. Taking again the example as support, the standard convolution use a-hundred and fifty-six kernel of dimension $5 \times 5 \times 3$ that move 8×8 times. That correspond to a number of multiplications in amount of: $256 \times 3 \times 5 \times 5 \times 8 \times 8 = 1228800$ The Depthwise convolution use three kernels of dimension $5 \times 5 \times 1$ that move 8×8 times. That correspond to a number of multiplications in amount of: $3 \times 5 \times 5 \times 8 \times 8 = 4800$ Instead, the Pointwise convolution use a-hundred and fifty-six kernels of dimension $1 \times 1 \times 3$ that move 8×8 times. That correspond to a number of multiplications in amount of: $256 \times 1 \times 1 \times 3 \times 8 \times 8 = 49152$ So adding them up together the results is a total of: 53952 multiplications. 53952 is a lot less than 1228800. With less computations, the network is able to process more in a shorter amount of time. Intuitively, the main difference between the two convolutions is that, in the normal convolution, the image is transformed a-hundredand fifty-six times. And every transformation uses up $5 \times 5 \times 3 \times 8 \times 8 = 4800$ multiplications. In the separable convolution, the image is really transformed only once in the Depthwise convolution. Then, the transformed image is simply elongated to a-hundred and fifty-six channels. Without having to transform the image over and over again, it is possible to save up computational power.

But there are disadvantages using Depthwise Separable convolution, because it reduces the number of parameters in a convolution. If the network is already small, the final model might end up with too few parameters and the network might fail to properly learn during training. If used properly, however, it manages to enhance efficiency without significantly reducing effectiveness, which makes it a quite popular choice.

3.4 Mobilenet V1

To deeply understand Mobilenet V2 model, first it is important to know well Mobilenet V1. Mobilenet (V1) [28] [74] is the first model to introduce Dephtwise Separable covolution. It is a classification network (not a detection one) and its reputation of light network grew up in the mobile application field in the last years and became the base for others light models.

As illustrated in Tab. 3.1 is it composed by:

- fourteen convolutional blocks;
- three final layers;

The default input image size for Mobilenet V1 (and V2) is 224×224 pixels, but there is also the possibility to use as input size 300×300 pixels. This second size works better in the case Mobilenet is used as backbone network of SSD (that have a defaults input size of 300×300 pixels). The first block is a standard convolution, while the following thirteen are a Depthwise

Input size: 224×224×3			
Block	**Layer**	**Filter, Stride**	**Output size**
block 1	conv	3×3, s2	112×112×32
block 2	conv dw	3×3, s1	112×112×32
	conv	1×1, s1	112×112×64
block 3	conv dw	3×3, s2	56×56×64
	conv	1×1, s1	56×56×128
block 4	conv dw	3×3, s1	56×56×128
	conv	1×1, s1	56×56×128
block 5	conv dw	3×3, s2	28×28×128
	conv	1×1, s1	28×28×256
block 6	conv dw	3×3, s1	28×28×256
	conv	1×1, s1	28×28×256
block 7	conv dw	3×3, s2	14×14×128
	conv	1×1, s1	14×14×512
block 8-12	conv dw	3×3, s1	14×14×512
	conv	1×1, s1	14×14×512
block 13	conv dw	3×3, s2	7×7×512
	conv	1×1, s1	7×7×1024
block 14	conv dw	3×3, s2	7×7×1024
	conv	1×1, s1	7×7×1024
classification block	avg pool	7×7, s1	1×1×1024
	FC	1024×1000, s1	1×1×1000
	Softmax	Classifier, s1	1×1×?

Table 3.1: Architecture of Mobilenet V1 [28].

Separable convolution with different convolutional kernels. Finally the last three layers present:

- an Average Pooling layer, to reduce variance, computation complexity and extract low level features from neighbourhood pixels;
- the Fully Connected layer, to transform the last features map in a vector describing the classification;
- the Softmax layer, to delete useless and bad classifications, giving in output only the best ones.

In Fig. 3.5 it is possible to see how a single block works. Each convolution (standard or depthwise ones) is followed by a batch normalization, that helps to normalize each output and by the ReLU6, nowadays the best activation function thank to its non-linearity property. A block is formed by a Depthwise convolution followed by a Pointwise ones (Fig. 3.5), that as reported previously generate a Depthwise Separable convolution. This network is designed to be small and to work well with the Depthwise Separable convolution. Indeed, as illustrated in Fig. 3.2, using the standard convolution improves the accuracy only of 1 mAP, but it increases a lot the weight of the network. In addition Mobilenet introduce two parameters to the network:

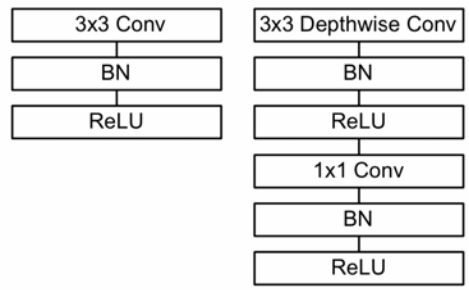


Figure 3.5: Mobilenet V1 block composed by a standard convolution (on the left) followed by a Depthwise Separable one (on the right) [28].

Model	mAP	MAdds	Parameters
Mobilenet (Conv)	71.7	4866M	29.3M
Mobilenet (Conv dw)	70.6	569M	4.2M

Table 3.2: Comparison of Mobilenet V1 with and without Separable convolution, in terms of accuracy on Imagenet Dataset [28].

- Width Multiplier α
- Resolution Multiplier ρ

The **Width Multiplier** control the input width of the layers, which makes M become αM, where α is between 0 to 1, with typical settings of 1, 0.75, 0.5 and 0.25. It consequently reduce the cost of the Depthwise Separable convolution: it impact with parameter α in the Depthwise convolution and with parameter α^2 in the Pointwise one. As shown in Tab. 3.3, decreasing α from 1 to 0.25 the accuracy drop off smoothly.

Model (Width Multiplier)	mAP	MAdds	Params
1.0 Mobilenet-224	70.6	569M	4.2M
0.75 Mobilenet-224	68.4	325M	2.6M
0.5 Mobilenet-224	63.7	149M	1.3M
0.25 Mobilenet-224	50.6	41M	0.5M

Table 3.3: Comparison of Mobilenet V1 at different α value on Imagenet dataset [28].

The **Resolution Multiplier**, control the input image resolution of the network through the parameter ρ^2, where ρ is between 0 to 1. It impact on the cost of the Separable convolution quadratically (with parameter ρ^2). Also in this case, the accuracy drops off smoothly across resolution from 224 to 128 (as illustrated in Tab. 3.4).

3.5 Mobilenet V2

Mobilenet V2 [66] [27] is the update of Mobilenet V1. It introduce a new convolutional block called "Inverted Bottleneck Residual Block" (Fig. 3.6). The two novelty of this block are:

- the bottleneck;

Model (Width Multiplier)	mAP	MAdds	Params
1.0 Mobilenet-224	70.6	569M	4.2M
1.0 Mobilenet-192	69.1	418M	4.2M
1.0 Mobilenet-160	67.2	290M	4.2M
1.0 Mobilenet-128	64.4	186M	4.2M

Table 3.4: Comparison of Mobilenet V1 at different ρ value on Imagenet dataset [28].

- the residual connection.

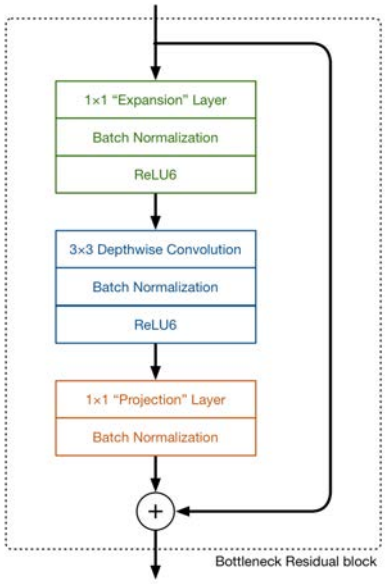


Figure 3.6: Mobilenet V2 "Bottleneck Residual block" [66].

The **bottleneck block** includes three convolutional layers this time. The last two are the same of Mobilenet V1: a Depthwise convolution that filters the inputs, followed by a 1×1 Pointwise convolution layer. However, the 1×1 layer now has a different function. In V1 the Pointwise convolution doubles or takes the same number of channels from the input image. In V2 is the opposite: it decreases the number of channels. This layer is called Projection layer, because it projects data with a high depth dimension (channels) into a tensor with a much lower one. For example, suppose to have a tensor with 144 channels, the Projection layer will reduce them to only 24 channels. This layer is also called a bottleneck layer because it reduces the amount of data that flows through the network. The first layer of the block is the new entry. It is again a 1×1 convolution and its purpose is to expand the number of channels of the input before it arrives to the Depthwise convolution. Hence, this layer called Expansion layer, produce an output depth dimension higher then the input ones. It is doing the opposite work of the Projection layer. The decision of how much expand the input is left to the Expansion factor. It is another hyper-parameters to choose during experiments to mitigate the trade-off between speed and accuracy. The default expansion factor is 6. So, the input and the output of the block are low-dimensional tensors, while the filtering step that happens inside block is done on

a high-dimensional tensor. Think at the low-dimensional data flowing between the blocks, like a compressed version of the real data. In order to run filters over this data, it is needed to uncompress them first. The expansion layer acts as an decompressor that unzip the data, then the Depthwise layer performs the filtering, and finally the Projection layer compresses the data to make it small again. The trick that makes this all work is that the expansions and projections are done using convolutional layers with learnable parameters, and the model is able to learn how to best compress and decompress the data at each stage in the network.

The second novelty about MobileNet V2 is the **residual connection** of the block. It has the function to help with the flow of gradients through the network. It make a sum operations between the input (residual) and the output features maps of the block. The residual connection is used only when the number of channels in input in the block is the same of the number of channels in output. This fact happen in some layers, while in all the others the number of channels increase (Tab. 3.5) and in these cases it is not possible to use the residual connection. As Mobilenet V1, each convolutional layer is followed by a batch normalization and by the activation function (again ReLU6). However, the output of the Projection layer does not have an activation function applied to it. This choice because this layer produces low-dimensional data and the authors of the Mobilenet V2 paper found that using a non-linearity function after this layer, destroy useful information. As is usual for this kind of model, the number of channels increases layer by layer, while the spatial dimension halved. But in this network the tensors remain relatively small, thanks to the bottleneck layers (Mobilenet V1 for example have a much larger tensor). Using low-dimensional tensors is the key to reduce the computations. Indeed, if the tensor is smaller, then fewer multiplications are needed in the convolutional layers. Tab.

Input size: 224×224×3				
Block	**Layer**	**Stride**	**Expansion Factor**	**Output size**
block 1	conv	s2	-	112×112×32
block 2	bottleneck	s1	1	112×112×16
block 3-4	bottleneck	s2	6	56×56×24
block 5-7	bottleneck	s2	6	28×28×32
block 8-11	bottleneck	s2	6	14×14×64
block 12-14	bottleneck	s1	6	14×14×96
block 14-16	bottleneck	s2	6	7×7×160
block 17	bottleneck	s1	6	7×7×320
classification block	conv pw	s1	-	7×7×1280
	avg poll 7×7	s1	-	1×1×1280
	conv pw	s1	-	1×1×?

Table 3.5: Mobilenet V2 architecture [66]: each line describes a sequence of 1 or more identical (modulo stride) layers. All layers in the same sequence have the same number of output channels. The first layer of each sequence use the stride reported and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor is always applied to the input size.

3.6 report the performance in terms of accuracy and network weight, of Mobilenet V1 and V2. Mobilenet V2 result a bit better in terms of accuracy, and there is an important improvement for the model weight. Mobilenet V2 take all the functions, structure, and strategy of Mobilenet V1. So, it use the same classification layers, it use Width and Resolution Multipliers and the same default input size.

Model	mAP	Params	MAdds	CPU time
Mobilenet V1	70.6	4.2M	575M	113 ms
Mobilenet V2	72.0	3.4M	300M	75 ms

Table 3.6: Performances in term of accuracy, weight and speed between the Mobilenet V1 and Mobilenet V2 [66]

3.6 Single Shot Detection (SSD)

Single Shot Detection (SSD) [49] [75] [31] is an object detection network belongs to the single-staged based networks. Its backbone network in the original paper is VGG16 [69], but it is possible to adapt SSD with different backbones, to achieve different goals. The author provide SSD with two possible input resolution: 300×300 pixels or 500×500 that correspond to the input image resolution. Thank to its versatility and the good performances with low resolution inputs, many researchers have used SSD in the last years, and also if the last version of YOLO (version 3) reaches best performance in terms of speed and accuracy, it probably still be used for a long time. As said in 2 SSD could be divided in two parts (Fig. 3.12):

- the backbone network (VGG16), that extract the features from the input;
- the front-end part that classify and localize the objects from the features provided.

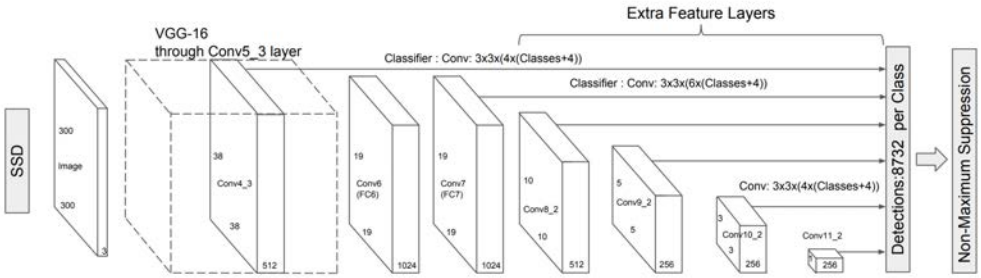


Figure 3.7: Architectures of SSD [49].

Cause in this thesis project we are not going to use SSD its original version, we will only describe the front-end part and its features, without touching the VGG16 network. Furthermore, we are going to use the SSD version with input resolution of 300×300, because a lighter input bring to lighter computation. Two are the main features of SSD:

1. multi-scale features;
2. default boxes.

Multi-scale features, is a technique using multiple layers to detect objects independently. In the backbone network the spatial dimension of the data gradually decrease (as common in this type of model) (Fig. 3.7), and also the resolution of the features map decrease with it. The front-end part use multiple features maps (arriving from the lasts layers of the backbone

network) to produce a predictions. It take different features maps to detect objects of different sizes. For example, it uses a feature map of dimension 4×4 to detect objects of large scale and a 8×8 feature maps to detect smaller objects (as illustrated in the Fig. 3.8). Multi-scale feature

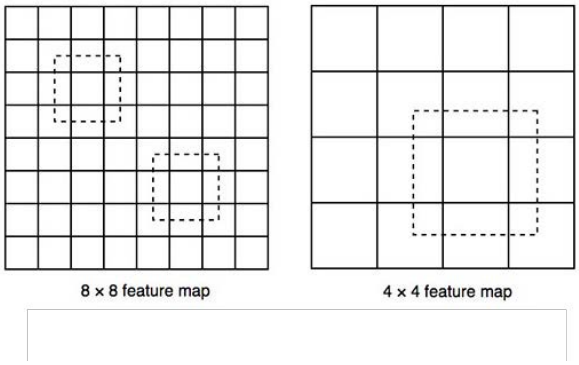


Figure 3.8: Different features maps scale to detect different scale object [49].

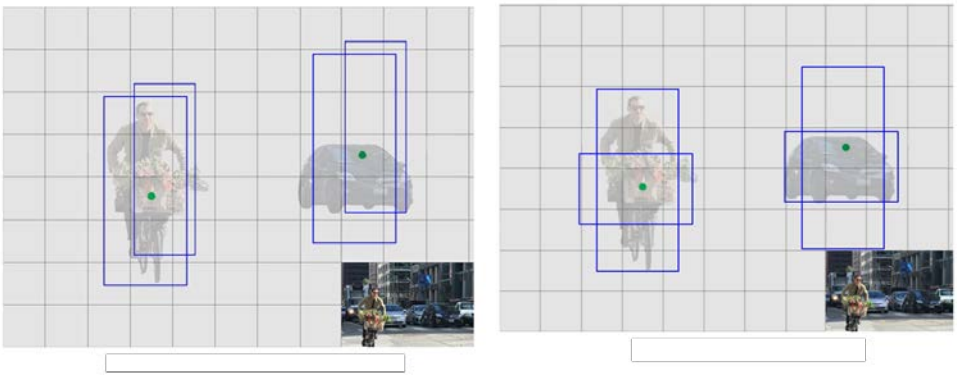
maps improve accuracy significantly. In Tab. 3.7 is reported the accuracy using different number of feature map layers.

Prediction source layer from:						**mAP** use bounding boxes?		# **Boxes**
38×38	19×19	10×10	5×5	3×3	1×1	Yes	No	
✓	✓	✓	✓	✓	✓	74.3	63.4	8732
✓	✓	✓				70.7	69.2	9864
	✓					62.4	64	8664

Table 3.7: SSD performance using different numbers of feature maps [49].

Defaults boxes The default bounding boxes are the equivalent to anchors in the more famous Faster R-CNN. It is possible to see, from the statistics of some database like KITTI [55], that the boundary boxes shapes are not arbitrary. Cars have have similar shapes each others and pedestrians have an approximate aspect ratio of 0.41. So, the boundary shape could be divided in cluster that are represented by default bounding box. So, instead of making random guesses to locate objects, it is possible to start the guesses based on those default boxes. A certain bounding box shape could work well for a certain object, but not for another one (Fig. 3.9a). So, it is important to cover a wide range of different shapes (Fig. 3.9b) Computing the default boxes in the algorithm, increase the complexity of the network and the time for training. So, SSD boxes are computed manually and pre-selected in a way to cover as most objects as possible. The algorithm use a fixed number of bounding boxes per layer (4 or 6) and produce one prediction from each box. Each features map share the same bounding boxes, but different layers can have different bonding boxes and in a different amounts. SSD for the bounding box predictions, instead of using global coordinates for the box location, uses the relative ones to the default bounding boxes (Δcx, Δcy, w, h).

Matching Strategies SSD predictions are classified as positive matches or negative matches. It uses only the positive matches to compute the localization cost (the mismatch between the true bounding box and the predicted one). SSD use the Intercept over Union (IoU) technique. It is the ration between the intersected area over the joined area of the two regions. In this case



(a) (b)

Figure 3.9: Different shape of labels for different objects: vertical shape for people and horizontal for cars [49].

the two regions are the default bounding box area (not the predicted boundary box one) and the true area. If the corresponding default bounding box has an IoU greater than 0.5 with the ground truth, the match is positive. For example, suppose to have three default bounding boxes. Only the first and the second one have an IoU greater then 0.5 (positive matches). Starting from the default bounding boxes it is possible to compute the localization cost of the correspondent predicted bounding boxes. This technique encourage to predict bounding box shapes close to the default one.

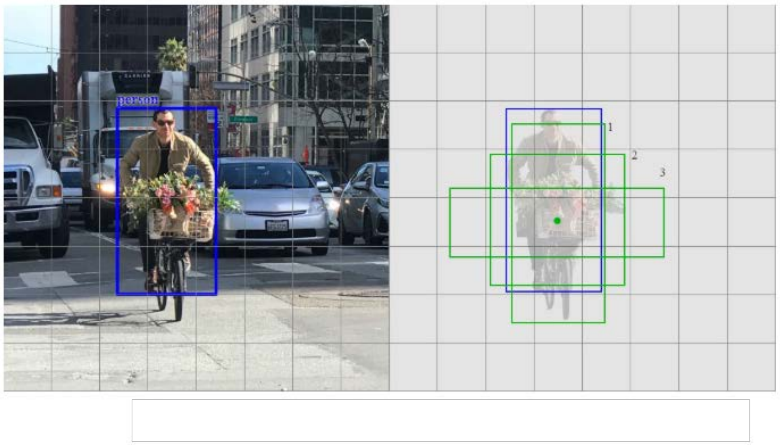


Figure 3.10: SSD different bounding box shape applied to the same cell of the grid [49].

Loss Function The loss function used by SSD is:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{3.1}$$

It consists in two terms: confidence loss (L_{conf}) and localization loss (L_{loc}), where N is the matched (positive) default boxes. SSD only penalizes predictions from positive matches. L_{loc} is

the smooth L1 loss between the predicted box and the ground-truth box parameters, i.e. it is the mismatch between them. The definition of the smooth L1 loss is:

$$L_{1,smooth} = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \tag{3.2}$$

and the formula of the L_{loc} is:

$$L_{loc}(x, l, g) = \sum_{i \in P_{os}}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} smooth_{L1}(l_i^m - \widehat{g}_j^m)$$

$$\widehat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \widehat{g}_j^{cy} = (g^{cy} - d_i^{cy})/d_i^h$$

$$\widehat{g}_j^w = log(\frac{g_j^w}{d_i^w}) \qquad \widehat{g}_j^h = log(\frac{g_j^h}{d_i^h})$$

where: l is the predicted box, g is the ground-truth box, cx and cy are the offsets for the center point, and w and h are the width and height of the bounding box.

The confidence loss is the softmax loss over multiple classes confidences. It represent the loss a class prediction. For every positive match, it penalizes the loss according to the confidence score of the corresponding class. For negative matches, it penalizes the loss according to the confidence score of the class "0" (no object detected or background class).

$$L_{conf}(x, c) = -\sum_{i \in P_{os}}^{N} x_{ij}^p log(\widehat{c}_i^p) - \sum_{i \in N_{eg}} log(\widehat{c}_i^0)$$

$$\widehat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)}$$

where: c is the confidences over multiple classes, xij^p is an indicator for matching i-th default box to the j-th ground truth box of category p.

Because the network is doing much more predictions than the real number of object in the image, there are more negative matches than positive matches. These imbalanced can produce bad results during training and the model learn better to recognize background instead of objects. To solve the issue SSD picks the negatives with the top loss and makes sure the ratio between the picked negatives and positives is at most 3:1. This leads to a faster and more stable training.

3.7 Single Shot Detection Lite (SSD Lite)

SSD Lite is lite version of the standard SSD. It uses as backbone network Mobilenet v2 and introduce the Depthwise Separable convolution also in the front-end part. SSD Lite take most of the setting and features from its father SSD. So, most of the previously described structures and techniques are still valid. All SSD convolution are substituted with Depthwise Separable convolution of the same stride and padding. As it is possible to see from Fig. 3.7, from Mobilenet v2 are discarded the last three layers (the ones needed to produce the classification) while remain the body composed by the bottleneck blocks. The connection between the two part is descripted in the following paragraph. As said in the previous section, SSD need a group of features maps to perform the multi-scale features detection. SSD Lite use a batch of six features maps (as in

SSD). The first two features maps are chosen from Mobilenet V2. They are the output of the block 13 and output of the last block (block 17). We choose these two layers because they are the proposed one from Mobilenet V2 [66] author. From the second features map (output of block 17) are generated four more features maps passing through four convolutional layers. The four features maps will be the remaining input for the multi.scale features detection. The six final features amps are:

- **Features map 1**: coming from *block*13, before be processed from for the multi-scale detection, it is normalized using an L2 normalization. The features map dimension is: $19 \times 19 \times 1576$.
- **Features map 2**: coming from *block*17 and has size of $10 \times 10 \times 1280$.
- **features map 3**: it is generated applying a Depthwise Separable convolution to the features map from *block*13. Its dimensions are: $5 \times 5 \times 512$.
- **Features map 4**: with the same pipeline used for feature map 3, it is generated processing the same. The dimension are $3 \times 3 \times 256$.
- **Features map 5**: it is generated, again in the same manner as the previous, and it has dimension $2 \times 2 \times 256$.
- **Features map 6**: finally, it generated from features map 5 and it is the last one. Its dimension is $1 \times 1 \times 128$.

The settings used by the detection layers are the same of the original SSD, to highlight:

- Mobilenet V2 uses an input size of 300×300 pixels, with the Resolution and the Width multipliers equal to 1.
- the number of default boxes for each features map, respectively of: 4, 6, 6, 6, 4, 4.
- the scales of each defaults bounding box that are: 0.07, 0.15, 0.33, 0.51, 0.69, 0.87, 1.05. This scales are computed to fit well with the OpenImage v4 Dataset [39], that is later described.

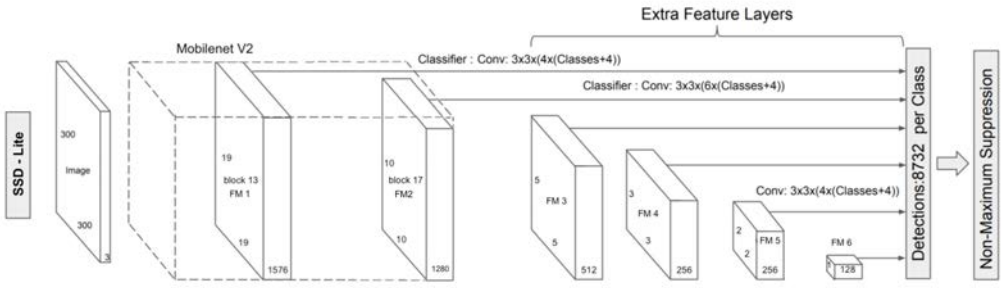


Figure 3.11: SSD-Lite architecture.

To give a better understanding of how this network works, in the following paragraph is reported the inference pipeline of an image of size 300×300. A single prediction is an array containing:

- the confidence predicted for each class (if the number of object class is 17, we have 18 classes, because we add also the nothing class)
- the bounding box coordinates of the best confidence (4 numbers).

So, if the number of class objects are seventeen, the prediction produce an array of size: $17 + 1 + 4 = 22$ elements. An inference procedure except:

1. The input image pass through the features extractor (Mobilenet V2) and produce the first two features map.
2. Following, the features map produced by the last block of the Mobilenet V2, pass through four more convolutional blocks to produce four more features map.
3. At each of these six features maps, is applied a grid. At the center of each cell of the grid we apply a number of predefined bounding box (four for features maps 1, 2 and 6 and six for features maps 3, 4 and 5), and from each bounding box we produced one prediction.
4. All the predictions are finally combined and analyzed: prediction of the same class are compared and in case grouped, prediction of different class in the same location are checked. Other operations are computed to arrive to a final set of predictions (No-Maximal-Suppression is performed to delete double bounding box or false ones).

3.8 DenseNet

To better understand DSOD and Tiny-DSOD, it is reported a brew summary about DenseNet. DenseNet in turn is based on ResNet [26] that first use a more deep convolutional network to reach better accuracy. As illustrated in Fig. 3.12, the most important elements of DenseNet are:

- the Dense Block;
- the Transaction Layer.

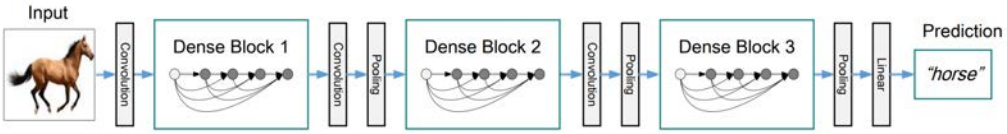


Figure 3.12: DenseNet schema, with Dense Blocks followed by Transition layers (Convolution and Pooling) [29].

The **Dense Block** is a group of convolutional layers deeply connected together. Indeed, as it is possible to see from Fig. 3.13, each layer take in input more then only one features map. The features maps are concatenated together and thanks to this the network can be thinner and compact. This bring to a more deep network, able to learn complex features and to increase the accuracy, but also to an efficient network in terms of memory and performance.

The **Transaction Layer** connects together the different Dense Block. It is made by a 1×1 convolution followed by 2×2 average pooling between two contiguous Dense blocks. Feature map sizes are the same in input and output in the Dense block, so that they can be concatenated together easily. DenseNet have several advantages:

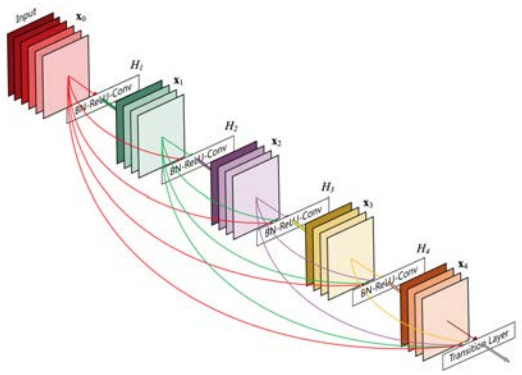


Figure 3.13: DenseNet Dense Block schema.

- it partially resolve the vanishing-gradient problem;
- strengthen feature propagation, encourage feature reuse;
- substantially reduce the number of network parameters.

3.9 DSOD

Deeply Supervised Object Detector (DSOD) [68] [80], an alternative of the original Single Shot MultiBox Detector (SSD), attempts to reach a better trade-off between speed and accuracy then SSD. DSOD use SSD as framework, it is a single-stage based model not using a region proposal system, and use as backbone network DenseNet [29].

Taking the advantage of DenseNet [29], DSOD introduce some new elements to increase the performance and the stability of the network. The main novelties are:

- **Deep Supervision (DP)** is a strategy to tackle the issue of the vanishing gradient. In many deep networks, can succeed that the flow of the gradient arrive to a point, where the variation start to be more and more small. So, the gradient doesn't move from the local value and doesn't provide an useful update. The cause is related to the maths under the calculation of the gradient (the partial derivatives). Basically they are a multiplication operations, and if two number with value less then one are multiplied together the results is a number minor of the previous two (e.g. $0.9 \times 0.8 = 0.72$). So, the gradient can became enough small that the model cannot learn anything more. The idea of DS is to bring the loss function, typically attached to the top part of the network, closer to different layers of the same. This allows each layer to use a less âĂIJdilutedâĂİ gradient to learn.

- **Transition without Pooling**, is a strategy that plans to connect the Dense Block without using the 2×2 average pooling (as succeed in DenseNet). So, it remain the Batch Normalization layer followed by a 1×1 convolution. This technique permits to increase the depth of the network without adding new layers in the Dense Block. This was not possible in DenseNet where, it cannot increase the number of Dense Block without changes the output size.

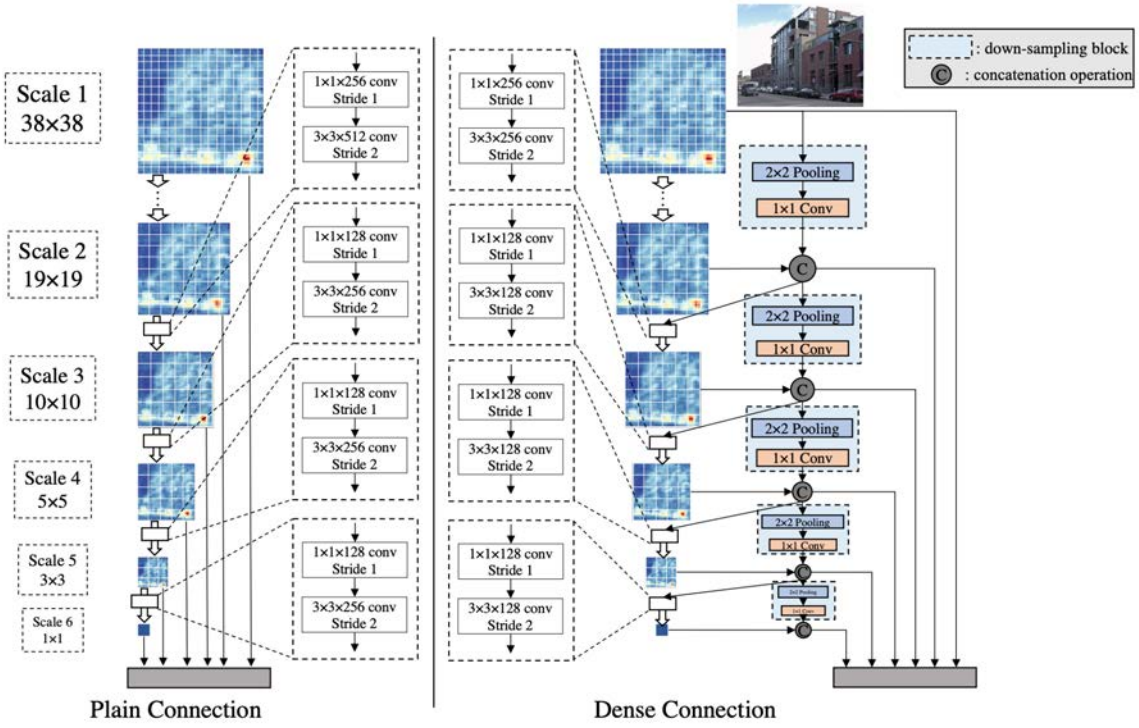


Figure 3.14: Comparison of SSD schema on the left and DSOD schema on the right [68].

- **Stem block** is used to modify the original DenseNet architecture. Instead of a using a 7×7 convolution layer with stride 2 followed by a 3×3 MaxPooling operation with stride 2, DSOD uses a stack of 3×3 convolution layers followed by a 2×2 MaxPooling. The first convolution layer has stride 2 whereas the others use stride 1. This to minimise information loss from the input image, because smaller filter sizes and strides tend to preserve information.

- **Dense Prediction Structure** help to improve the detection accuracy. It concatenates one-to-one the features map in output from the previous layer, with the down-sampled high-resolution features map arriving in output from the actual layer. This strategy works well because the high-resolution features map preserve spatial information, while the features map of the previous layer have useful information for classification. It is possible to see the difference in respect to SSD plain approach from the Fig. 3.14. The down-sampling operation is done using a 2×2 max pooling layer with stride 2 followed by a 1×1 convolution layer with stride 1.

In Tab. 3.8 is reported the architecture of DSOD.

Input size: 300×300×3					
Block		**Layer**	**Filter, Stride**		**Output size**
Steam	block 1	conv	3×3, s2		150×150×64
	block 2	conv	3×3, s1		150×150×64
	block 3	conv	3×3, s1		150×150×128
	block 4	max pool	2×2, s2		75×75×128
dense block 1	block 5	conv	1×1, s1	× 6	75×75×416
		conv	3×3, s1		
transition block 1	block 6	conv	1×1, s1		75×75×416
		max pool	2×2, s2		38×38×416
dense block 2	block 7	conv	1×1, s1	× 8	38×38×800
		conv	3×3, s1		
transition block 2	block 8	conv	1×1, s1		38×38×800
		max pool	2×2, s2		19×19×800
dense block 3	block 9	conv	1×1, s1	× 8	19×19×1184
		conv	3×3, s1		
transition block 3	block 10	conv	1×1, s1		19×19×1184
dense block 4	block 11	conv	1×1, s1	× 8	19×19×1568
		conv	3×3, s1		
transition block 4	block 12	conv	1×1, s1		19×19×1568
DSOD prediction layers					

Table 3.8: Architecture of DSOD.

3.10 Tiny-DSOD

Tiny-DSOD [43] is an object detection network based on DSOD. It promises good performances in mobile devices and try to maintain a good trade-off between speed and accuracy. The main novelty proposed by Tiny-DSOD are:

- the Depthwise Dense Block (DDB) based backbone;
- the feature-pyramid-network (D-FPN) based front-end.

The **Depthwise Dense Block (DDB)** is an efficient network structure to combine Depthwise Separable convolution with densely connected networks (DenseNet). Tiny-DSOD provide two different DDB. As shown in Fig. 3.15, it is easy to understand that the first version (Fig. 3.15a) is based on the Inverted Residual Bottleneck proposed by Mobilenet V2 [66]. It first expands the input by a factor w (using 1×1 convolution), generating a features map of $n \times w$, where n is the number of channel (depth dimension) of the input, and w is an hyper-parameter that can control the capacity of the network. Then the Depthwise convolution is applied, and the features map is projected to g channels (using another 1×1 convolution), where the g is the growth rate of the block. Finally the input and output features maps are concatenated together (not added as in the Mobilenet V2 model). This block has two hyper-parameters w and g.

The paper author find out two important weaknesses in this block:

- the complexity of a series of L blocks is $O(L^3g^2)$. So, the computation increase rapidly with L, and if a small g is used, it will hurt the accuracy of the results.
- there is a redundancy when two Dense Block are concatenated, because there is a Pointwise

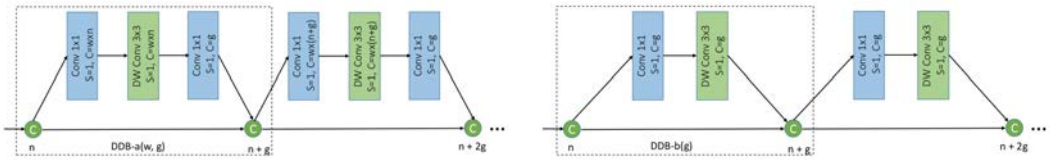


Figure 3.15: Tiny-DSOD Depthwise dense blocks (DDB). In the rectangle, "S" means the stride of convolution, and "C" means the number of output channels. Numbers under the concatenating node (green C with circle) means the number of output channels after concatenation. (a) is stacked DDB-a parameterized by growth rate g and expand ratio w. (b) is stacked DDB-b parameterized by growth rate g [43].

convolution at the end of the first block and another one at the beginning of the following block.

To overcome these weaknesses the author provides a second Dense block (Fig. 3.15b). The input features map is first compressed to g channels and then the Depthwise convolution is performed. The generated output is concatenated to the input without the Pointwise convolution. The overall complexity becames $O(L^2g^2)$. Also the experiments made in the paper verify that this block perform better then the previous one. The final architecture of the backbone network is shown in Tab. 3.9.

Input size: 300×300×3				
Block		**Layer**	**Filter, Stride**	**Output size**
Steam	block 1	conv	3×3, s2	150×150×64
	block 2	conv	1×1, s1	150×150×64
	block 3	conv dw	3×3, s1	150×150×64
	block 4	conv	1×1, s1	150×150×128
	block 5	conv dw	3×3, s1	150×150×128
	block 6	max pool	2×2, s2	75×75×128
Dense stage 0	block 7	DDB(32)×4		75×75×256
Transition layer 0	block 8	conv	1×1, s1	38×38×128
	block 9	max pool	2×2, s2	
Dense stage 1	block 10	DDB(48)×6		38×38×416
Transition layer 1	block 11	conv	1×1, s1	19×19×128
	block 12	max pool	2×2, s2	
Dense stage 2	block 13	DDB(64)×6		19×19×512
Transition layer 2	block 14	conv	1×1, s1	19×19×256
Dense stage 3	block 15	DDB(80)×6		19×19×736
Transition layer 3	block 16	conv	1×1, s1	19×19×64

Table 3.9: Tiny-DSOD backbone architecture.

The **feature-pyramid-network (D-FPN)** is a lightweight version of FPN [46], that fuse semantic information from neighborhood scales to speed up object detection and to overcome the weakness of SSD and DSOD fornt-end part, in terms of accuracy in the predictions. As shown in Fig. 3.16, the front-end predictor D-FPN consist in a down-sampling path and a

reverse up-sampling one. The reverse-path (up-sampling) has been demonstrated being very helpful for small object detection in many works [17], [46], [81]. But majority of this works use deconvolution to up-sample the features maps, and this technique increase the complexity of the model. To overcome this issue the paper propose a cost-efficient solution: as illustrated in Fig. 3.16, the features map is up-sampled using a Bilinear interpolation layer followed by a Depthwise convolution. The resulted feature maps are merged with the same-sized feature maps in the bottom layer via element-wise addition.

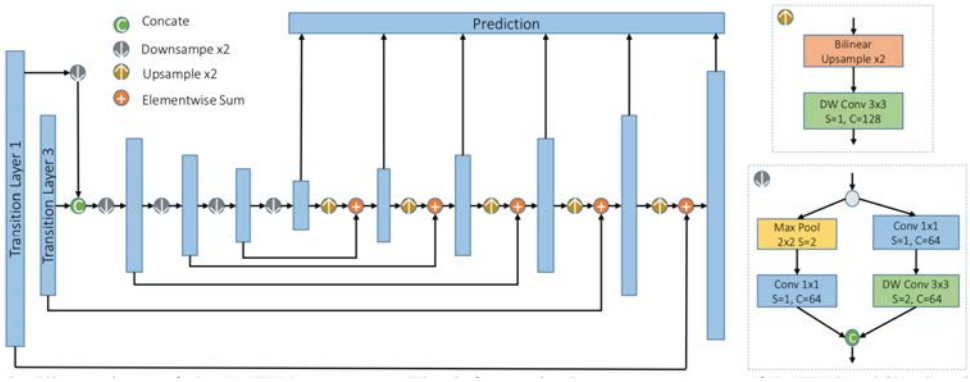


Figure 3.16: D-FPN structure: the left part is the over structure of D-FPN, while the right part further depicts the details of the up-sampling (top-right) and down-sampling (bottom right) modules in D-FPN. Note both sampling are by factor 2, "S" is the stride of convolution, and "C" is the number of output channels [43].

Because Tiny-DSOD use the SSD framework for the detection, they have in common the last layers of the network. They also use the same defaults box and same multi-scale features parameters. What really change are the six features maps used for the multi-scale detection and their generation. As it is possible to see from the schema at Fig. 3.16 each final feature map is the sum of two element, one arriving from the up-sampling and one from the down-sampling. The six features maps in input for the multi-scale detection are:

- **Features map 1**: it come from the down-sample of the features map generated from $block19$ (it is the only one not coming from a sum operation). The features map dimension is: $1 \times 1 \times 1024$.
- **Features map 2**: it is the sum of the features maps coming from the up-sampled $featuremap1$ and $block19$. The features map dimension is: $3 \times 3 \times 512$.
- **features map 3**: it is the sum of the features maps coming from the up-sampled $featuremap2$ and $block18$. The features map dimension is: $5 \times 5 \times 256$.
- **Features map 4**: it is the sum of the features maps coming from the up-sampled $featuremap3$ and $block17$. The features map dimension is: $10 \times 10 \times 128$.
- **Features map 5**: it is the sum of the features maps coming from the up-sampled $featuremap4$ and $transitionlayer3$ ($block16$ of the backbone network). The features map dimension is: $19 \times 19 \times 128$.
- **Features map 6**: it the sum of features maps coming from $transitionlayer1$ ($block12$ of the backbone network) and the up-sampled $featuremap5$. The features map dimension is: $38 \times 38 \times 128$.

Obviously, the inference pipeline is the same of the SSD one.

Chapter 4

Experiments

In this section we are first going to show the API library and tools, used for the experiments. Following we present the datasets used for the tests and the tests them-self. Finally we will show the results obtained.

4.1 Setup

4.1.1 Tensorflow Lite

Tensorflow Lite [72] [54] is TensorFlow's lightweight solution for mobile and embedded devices. It permits to run machine-learned models on mobile devices with low latency. We decide to integrate our networks in the Android application using this framework, because it brings many advantages. It is presently supported on Android and iOS via a C++ API, and it has also a Java Wrapper for Android Developers. Additionally, on Android devices that support it, the interpreter can also use the Android Neural Networks API for hardware acceleration, otherwise by default it uses the CPU for the execution. TensorFlow Lite includes a runtime on which you can run pre-existing models, and a suite of tools that you can use to prepare your models for mobile and embedded devices. It is not yet designed for training models. Instead, you can train the model on a higher powered machine, and then convert that model in .TFLITE format, which is finally loaded into a mobile interpreter. TensorFlow Lite is presently in developer preview, so it may not support all operations in all TensorFlow models. Despite this, it works with common Image Classification models, including Inception and MobileNet [57]. As illustrated in Fig. 4.1, the pipeline to produce a .TFLITE file is:

1. train the network using conventional strategies and frameworks of Tensorflow;

2. export the inference network and its weights in a checkpoint files; checkpoint is made by different files: .ckpt.data, .ckpt.meta and ckpt.index, that contain the model, its configuration and its weight.

3. freeze the exported model in a .pb file, using the script proposed by Tensorflow [72] or produce a personal script following the Tensorflow guideline; the freezing function permits to unify the model structure and its configuration (also the weights) in the .pb file;

4. convert the freezed network (.pb file) in a .TFLITE file, using the Tensorflow tool;

5. import the .TFLITE file in the Android application and using the Neural Network API Library to integrate it in the code.

There are two possible .TFLITE files to use:

- standard TFLITE file;
- quantized TFLITE file: post-training quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy. It is possible to perform this technique using an already-trained float TensorFlow model, in the conversion process from .pb file to .TFLITE format.

We chose to use the quantized TFLITE file for our experiments, because it permits to perform with an higher FPS, without visible drawback in accuracy.

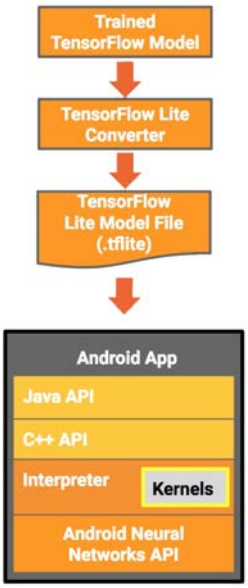


Figure 4.1: Tensorflow Lite pipeline to import a trained model in an Android application.

We decide to use Tensorflow lite in this thesis project, because:

- it fully supports Mobilenet, and has a basic API guide;
- it is the easiest way to import a network in an Android application;
- it performs well in term of efficiency and network optimization for light devices.

4.1.2 Keras

Keras is a high-level neural networks API, written in Python and capable to run on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

We choose Keras as main API because it has a wide range of function, useful for our project as:

- it allows easy and fast prototyping (through user friendliness, modularity, and extensibility);
- the library integrates the Mobilenet V1 and V2 networks;
- it is compatible with Tensorflow (in 2018 Keras belong to Tensorflow APIs) and many tools are provided to convert networks and weights from Keras to Tensorflow and viceversa;
- it runs seamlessly on CPU and GPU.
- it has an efficient optimization of the training process;
- it has a good documentations and an active community working on it.

4.1.3 Hardware

The hardware provided for the project consists in:

- an **Asus ZenFone AR** 4.2 equipped with: 4GB of RAM, 64-bit CPU Qualcomm Snapdragon Quad-Core 821 2.35 GHz, optimized for Tango, Adreno 530 GPU, 23 megapixel camera, and Android 8 installed. The whole specific are provided at [2];
- a desktop Computer with: GPU Nvidia Geforce GTX 1050ti 4GB, 16GB RAM, i7-4770 CPU 3.40GHz. Installed there is Linux 18 operative system;
- a GPU cluster with the possibility to use a Nvidia Geforce GTX 1080ti 12GB.

Figure 4.2: Asus Zenfone AR [2].

As said in chapter 1, the Asus ZenFone AR is equipped with a depth sensor. Finally, we dont't use this sensor for the object detection task. Indeed, all the networks we are analyzing, don't use depth images or they are not able to take information from them. Probably, the depth map will be useful for the navigation purpose, that is not part of this thesis work.

4.2 Software Structure

The code produced for the project development is written in Python 3.6, using the Keras API library and Tensorflow version 1.12. We are not using the last released Tensorflow 1.13, because of its incompatibility with some tools provided from the Object Detection section of Tensorflow. The code base is taken from a Github repository of Pierluigi Ferrari [12], who has developed SSD algorithm in Keras. He produces also some useful tools to make tests and inferences on the network. He follows the SSD paper to write the code and uses the test rules of Pascal VOC 2012 challenges (that could be considerate functional also for our purpose). Deeply modifying the code of SSD300, we obtain the two algorithms needed: SSD-Lite with Mobilenet V2 and Tiny-DSOD. The two algorithms follow the respective paper guidelines and parameters, trying to be as close as possible to the original models. More tools and scripts have been created and provided to help in training and testing, and also in the selection of useful images from dataset. The two networks share the majority of the code, beginning from the last layers of SSD300 used by both models to classify and localize objects from the features maps provided by each algorithm. We decide to write the two algorithms from scratch, instead of using some code proposed in the web, for three main reasons:

- to deeply understand these two algorithms, it is important to write down each of its part, otherwise it is impossible to completely understand the models and produce useful modifications;
- it is important to have the code in Keras or Tensorflow frameworks, because they are the only way to later export the models in Tensorflow Lite. E.g. Tiny-DSOD is provided in Caffe framework.
- to have the possibility to test the algorithms, without be affected by the framework or the language choice. We decide to share as much as possible code between the two models.

For the training process, of both the models, this parameters have been chosen: the number of epochs for the training are 120, with 943 iteration each and a batch size of 24 (maximum permitted from the GPU).

4.3 Datasets

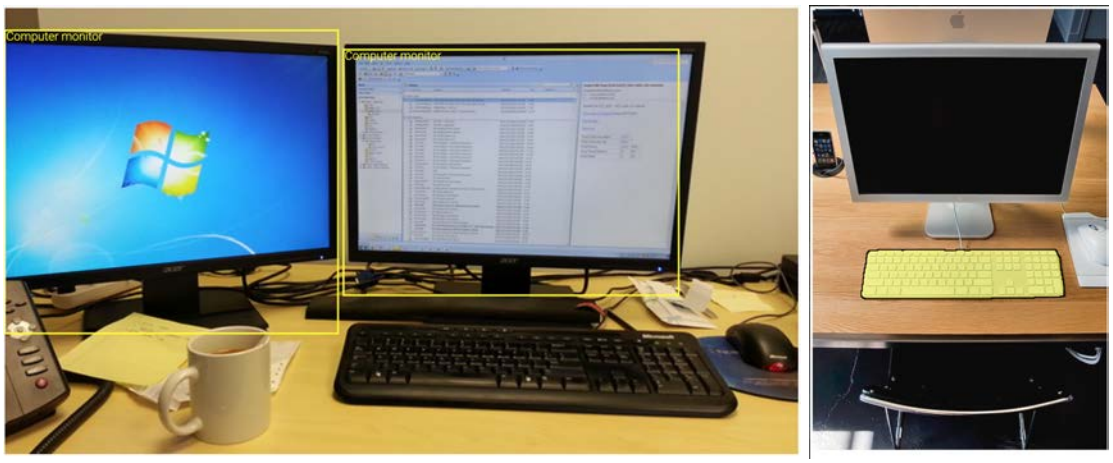
In this thesis work we are going to use two different datasets:

- **OpenImage V4 Dataset selection** [39]: to train and test the proposed models.
- **Office Dataset**: specifically created for this project and used to test the models.

4.3.1 Train Dataset

OpenImage V4 [39] [38] is a dataset of 9M images annotated with image-level labels, object bounding boxes, object segmentation masks, and visual relationships (Fig. 4.3). It contains a total of 16M bounding boxes for 600 object classes on 1.9M images, making it the largest existing dataset with object location annotations. The boxes have been largely manually drawn by professional annotators to ensure accuracy and consistency. The images are very diverse and often contain complex scenes with several objects (8.3 per image on average).

Having a large number of object categories, and specially, objects from office environment, it is a good dataset to train our models. We take only the images containing the objects useful for



(a) (b)

Figure 4.3: Examples of bounding box (a) and segmentation (b) in OpenImage V4 dataset. [38]

our training and the related labels. In Tab. 4.1 are reported the seventeen objects that represent an office environment in our project. They statistically are common objects in an office room and they are also possible objects that an impaired person would like to look for. As reported in 1, the project plans to detect objects, but can also navigate the user close to the possible position where the object may be located. Suppose to look for a PC monitor, the user is now located close to the door. The object detection network doesn't find any PC monitor for the moment, but it recognizes a desk. The application will advice the user to move in-front-of the desk, to see if it is able to find the PC monitor from the new position. This extra help, given by the application, is made thanks to a statistics generated on the OpenImage V4 dataset, where the positional relations are computed between different objects. This is not part of the thesis, but part of the project, and that to justify the presence of objects apparently useless, but that can be useful for the application to find other close items. In Tab. 4.1 it is possible to see also the number of objects for each single item (used for the training process) and the complexity of each object defined in a scale from 1 (easy) to 3 (very complex). It is important to know what objects are considered complex to detect and why. This rank is given without testing previously the models on these objects, but it will explains the behaviour of the detectors. Difficult object examples are:

- **chairs, backpack**, that are considered complex for their variance. Indeed, they have different shapes, colors and sizes. This bring to an undefined standard model for a chair and to an hard detection.
- **mouse, mug, light-switch** are considered complex for their size. Indeed they have a little size and they are difficult to be detected from far points of view.

Each object category has different numbers of training images. This fact could be useful, to understand which threshold, the network can use to learn well. We take a maximum number of 10.000 objects per category, to limit the dataset size. Not all the objects reach this threshold, and finally, we obtain a dataset with an amount of objects equal to 96742 in 43301 pictures. The selected images are equally divided in three groups:

- 24762 images for the training set ($\sim 60\%$);

- 9035 images for the validation set ($\sim 20\%$);
- 9504 images for the test set ($\sim 20\%$).

Using a script the objects are divided equally and homogeneously in the three sets (following the written percentage).

OpenImage V4 has also extra information about the appearance of the objects:

- occluded indicates that the object is occluded by another object in the image;
- **truncated** indicates that the object extends beyond the boundary of the image;
- **group** indicates that the box spans a group of objects (more than 5 instances which are heavily occluding each other and are physically touching) (e.g., a bed of flowers or a crowd of people);
- depicted indicates that the object is a depiction (e.g., a cartoon or drawing of the object, not a real physical instance);
- inside indicates a picture taken from the inside of the object (e.g., a car interior or inside of a building).

We decide to discard all the pictures that present objects with: truncation, occlusion, depiction or inside other objects. We do that because doing a pre-test on a training set with all the images and with a training set without the "complex" images, it highlighted the best accuracy of the second training set. Indeed, training the models with too complex images, can bring the detector to not define a good representation for an objects or to define a wrong one. It could be useful to have complex pictures in the training set, because it helps to detect objects in complex positions or partially visible. But it is important to control the number of "complex" image, to don't have too much of them, otherwise they hurt the training.

4.3.2 Test Dataset

There are two Test datasets because there are two aspects to take care of. First, it is important to understand if the training is gone well, and for this motif we use a selection of images from the OpenImage V4 dataset (as previously said the 20% of the selected images). Indeed, it comes from the same images distribution of the Train and Validation set, and thanks to this, it is possible to understand the weaknesses and the strengths of the networks in term of lean ability. Instead, using a test set coming from a different distribution, helps to understand how robust is the model. Additionally, it also highlights if the model learns the representation of a certain item, or learns it only for the specific distribution. Furthermore, using a real environment as Test set, the test returns a better response of how well the model works in the real-life. The Office Dataset is created on purpose for this project. Indeed, the pictures are taken from a real office of the University of Lincoln and it contains all the seventeen objects we are looking for (Fig. 4.1). The dataset is composed by pictures where only one object is labeled in each image. All the pictures are captured from the same environment, with the objects fixed in the same position. What change is the user point of view.

We take pictures from three different points of view (Fig. 4.4):

1. **door position**, when the user opens the door and looks inside the room office;
2. **guest position**, when the user sits down on the chair in front of the desk;

Object	# Images	Complexity
Backpack	1314	2
Book	10000	2
Bookcase	5708	1
Chair	10000	3
Desk	10000	1
Door	10000	1
Keyboard	4757	1
Lamp	3849	2
Laptop	10000	1
Light switch	114	3
Monitor	6384	1
Mouse	841	1
Mug	2403	1
Plant	10000	1
Telephone	316	2
Whiteboard	1056	1
Window	10000	1

Table 4.1: Selected object categories from OpenImage V4 dataset, with the number of images per object and the complexity each one.

Figure 4.4: Schema of the office setting, created to generate the Office dataset.

3. **owner position**, when the user sits down on the chair behind the desk.

Then for each position we take pictures using three different heights, representing the typical positions in which a user hold the smart-phone (a person of mean height of 175cm). The three heights are:

1. **pelvis height**, when user holds approximately the smartphone at the height of the pelvis;

2. **chest height**, when user holds approximately the smartphone at the height of the chest;

3. **head height**, when user holds approximately the smartphone at the height of the head.

Finally for each position and for each height, we take the pictures in three different inclinations of the smartphone:

1. **45°(right of the user)**, when the user holds the smartphone with a rotation of 45°on the right;

2. **portrait (0°)**, when the user holds the smartphone in portrait mode, without rotation;

3. **-45°(left of the user)**, when the user holds the smartphone with a rotation of 45°on the left.

We decide not to take pictures in the panorama mode, because it is uncommon or uncomfortable position to hold the smartphone when a person is using it. The only case where a user holds the smartphone in panorama mode, is when he takes pictures, and it is not our case. The total amount of captured pictures is:

$$3\,positions \times 3\,heights \times 3\,inclinations \times 17\,objects = 459\,pictures\ (27\,per\,object)$$

In Fig. 4.5 there are some picture examples from the Office Test set.

(a) (b) (c)

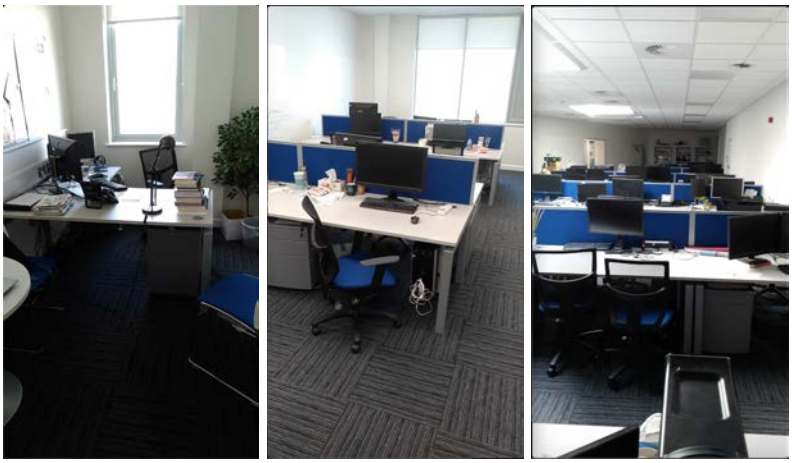
Figure 4.5: Three pictures taken from the Office Test set. The three images are centered on the desk object, respectively from: door position (a), guest position (b), and owner position (c).

The Office dataset has also the goal to provide a support for future experiments. Thanks to its setting other researchers can use it, to compare more algorithms for the same purpose.

4.3.3 Video Test Dataset

The Video Test set is composed by nine different videos, taken from three different office rooms:

- **little office** 4.6a of size $10m^2$ (it is the one from which we take the Test Dataset pictures);
- **medium office** 4.6b of size ($50m^2$) (it doesn't contain all the trained objects);
- **big office** 4.6c of size $300m^2$ (it doesn't contain all the trained objects).



(a) (b) (c)

Figure 4.6: Three office room of different dimensions: (a) little office of size $10m^2$, (b) medium office of size $50m^2$, and (c) big office of size $300m^2$.

We choose three different room dimensions, to better understand if the models are able to recognize objects and from what distance. It is important that a good algorithm is able to recognize evident and big object from long distance, to later navigates the user near them to find little or less evident objects. In each of the three rooms, three different videos are recorded at three different speeds. The speed is intended as the velocity of the movements that the user produces, while he is walking in the room. The different speeds are:

- **slow speed**, the walk and the movements of the user are slow, so the images are not too much blurred and the scene change gradually;
- **medium speed**, the walk of the user and the movements are a bit faster that the previous videos, so the frames start to blur and sometimes the scene change drastically;
- **high speed**, the user walk and movements are fast, many frames of the video are blurred and frequently the scene change drastically;

Changing the speed of the videos help in understanding how robust the models are with blurred frames and to see if the computational time of each frame is low enough.

We decide to record these nine videos, because it is important to understand how a model works in real-life. The videos don't provide a scientific test (the frame of the video are not labeled), but only a visual feedback that returns an idea about weaknesses and strengths of the networks.

4.4 Test pipeline

The test pipeline plans to:

1. test the **SSD-Lite** on the selection OpenImage V4 Test set, to obtain the accuracy in the detection of each object.
2. test the **SSD-Lite** on the Office Test set, to obtain the accuracy in the detection of each object.
3. test the **SSD-Lite** on the selection Video Office Testset, to obtain a visual feedback of how the model is working.
4. test the **Tiny-DSOD** on the selection OpenImage V4 Test set, to obtain the accuracy in the detection of each object.
5. test the **Tiny-DSOD** on the Office Test set, to obtain the accuracy in the detection of each object.
6. test the **Tiny-DSOD** on the selection Video Office Testset, to obtain a visual feedback of how the model is working.

These tests provide a good support to understand well what is the best algorithm for our project. We highlight again that we are looking for an algorithm with the following features:

- it have to work in real-time performances ($\geq 20FPS$);
- it have to be robust to anomalous movements of the user and to blurred frames;
- it have to avoid as many detection errors as possible: it is better to found zero objects instead a wrong one.

4.5 Results

As it is possible to see from the results on Tab. 4.2, the training of these two networks bring to low accuracy performance. But before give a final judgement, we will report why they reach these values, deeply analyzing each network and the results for each object.

The mean accuracy reveal that the best models with this training setting is SSD-Lite. Indeed, it reaches an higher mean accuracy (16.2 mAP) and also in the single objects, it quite always has better results. The only anomalous case is the Monitor detection, where SSD-Lite has a really bad accuracy. To well understand the value reached, we first remember (as reported in Tab. 4.3) that the accuracy of these algorithms in the COCO dataset is respectively: 22.1 mAP for SSD-Lite and 23.2 mAP for Tiny-DSOD. COCO dataset have a big amount of images (330K images) and a high number of object categories (80 object categories). Another possible comparison is the Pascal VOC 2012 dataset, where both the models reach really good results: SSD-Lite 68.0 mAP and Tiny-DSOD 72.1 mAP, and where the number of images is lower then COCO (11530 images) and the number of object categories is only 20. This is to say that really different results can be obtained using different training and test sets.

It is clear that Tiny-DSOD doesn't learn a lot from this training. Its mean accuracy is very low and the accuracy per object reveals that it learns to recognize really few of them: more then half have less then 10 AP accuracy, and five of them have less then 1 AP accuracy. Analysing some of the predictions (Fig. 4.7b) and 4.7d), it is possible to see, that Tiny-DSOD produce

Object	AP SSD-Lite		AP Tiny-DSOD	
	OpenImage	Office	OpenImage	Office
Backpack	15.5	0.0	3.2	0.0
Book	9.1	0.0	0.6	0.0
Bookcase	31.4	43.7	14.9	22.6
Chair	1.0	0.0	0.1	0.0
Desk	11.1	3.1	9.8	5.1
Door	17.0	18.2	9.7	8.4
Keyboard	18.5	0.2	3.3	0.0
Lamp	12.9	0	4.6	0.0
Laptop	22.4	19.6	10.0	11.1
Light-switch	0.0	0.0	0.0	0.0
Monitor	2.4	11.4	5.1	5.6
Mouse	15.3	0.0	4.5	0.0
Mug	28.8	0.0	5.4	0.0
Plant	24.3	60.3	16.4	32.7
Telephone	0.0	0.0	0.0	0.0
Whiteboard	64.7	7.3	19.3	3.9
Window	0.3	0.0	0.8	0.0
MEAN	**16.2**	**9.6**	**6.3**	**5.3**

Table 4.2: Results in terms of accuracy of SSD-Lite and Tiny-DSOD, both on OpenImage and Office test set.

Method	mAP			FPS	Params	FLOPS
	COCO	OID	Office			
SSD-Lite	22.1	16.2	9.6	~ 20	4.30M	0.8B
Tiny-DSOD	23.2	6.3	5.3	~ 20	0.95M	1.12B

Table 4.3: Results in tearms of mean accuracy (mAP), speed (FPS), and weight (Params and FLOPS) of SSd-Lite and Tiny-DSOD.

many wrong bounding boxes with high confidence. This prove the inconsistency of the results. We later confirm these performances also from the video analysis. Instead, SSD-Lite reaches a low, but acceptable, mean accuracy value. It is not able to recognize all the objects, but as it is possible to see from the predictions (Fig. 4.7a and 4.7c), it produces only few correct detection with a nice confidence. Obviously, it also produces sometimes wrong predictions. Also SSD-Lite has four objects with accuracy less then 1 AP, but more then half of the objects have an accuracy greater then 10 AP. A good threshold of confidence for SSD-Lite is of 70%. Using this threshold the majority of the wrong predictions are discarded and the majority of the right ones are taken. The main reasons, because of the low accuracy in the OpenImage dataset, are reported in the following list, for both SSD-Lite and Tiny-DSOD:

- the number of images for the training of some object categories are not enough. They are the case of the Light-switch, the Mouse and the Telephone categories (accuracy less then 1 AP), where the low number of training images (Tab. 4.1) doesn't permit to learn the

objects representations.

- The complexity of some objects, like the Chair, bring to the impossibility to learn their representations. Too many different image examples bring to an inconsistency in the learning of the object.
- The OpenImage dataset contains many complex scenes, that sometimes can bring to a difficult detection or to a difficult learn of the object representation. It is the case of little size objects (in the images where they cover a little area in respect to the total one of the image) or of the Book that can be found in different positions and many times grouped together.
- The total number of training images is not enough for the complexity of the networks. A complex network need more examples and a longer training to perform well. The high number of hyper-parameters to train asks for a huge amount of example images.
- The accuracy measures the correctness of the predictions, and also the precision of the location. So, a low accuracy is not necessarily an indicator of the impossibility to detect a certain object. Suppose that the model detects an object, but with a medium confidence (50% confidence). In this case it is important to look also at the accuracy of the wrong predictions. If they have all big gap with the correct predictions, adjusting the threshold, it is possible to discard them without discard the correct ones.

Tiny-DSOD is more affected from these weaknesses. As shown in Tab. 4.4 its number of parameters (Params) is very low and, because of that, it has less learning ability then SSD-Lite (that have four times its parameters). This is probably the main reason of the gap between the two models. Tiny-DSOD needs a longer training and with a major number of image examples. Instead, looking at the Office Test set results, we can see they are worse then the OpenImage ones. Apart the previously said motifs, in this case there are some new elements to highlight:

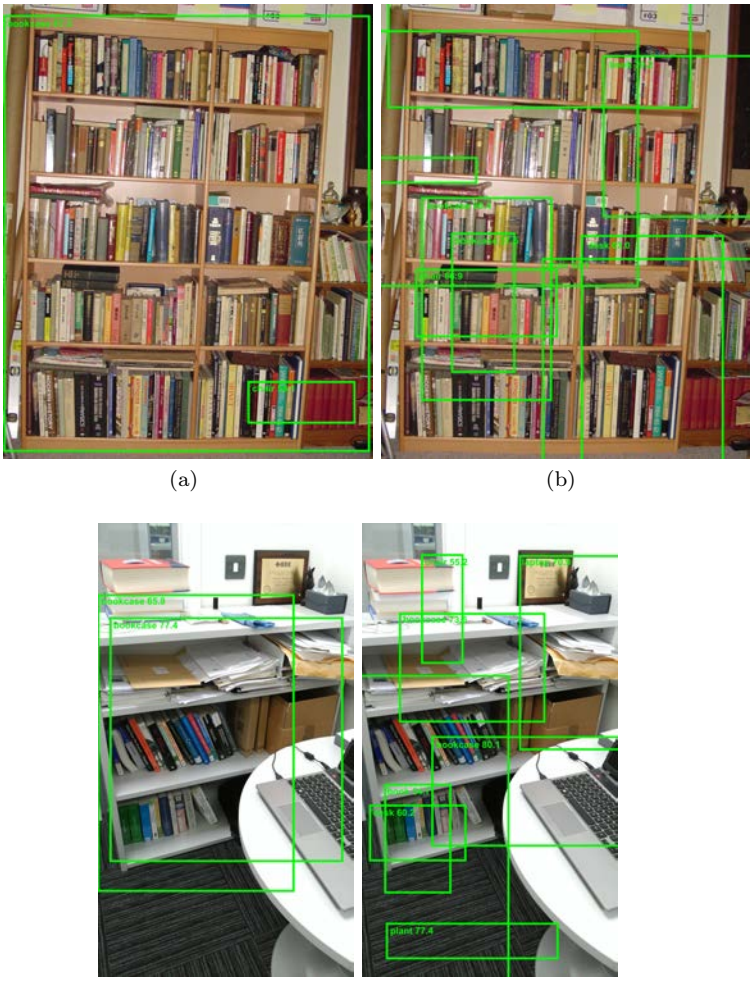
- the Office test set comes from a different distribution compared to the training set. Obviously, this fact decreases the accuracy, because it is more probable that the objects, the algorithms have to predict, are really different from the ones it learns from.
- There is a group of objects (as Mouse, Light-switch, and keyboard), that are always of little size in the test images. That because these objects never be in foreground and as close as needed to the camera.
- The dataset has only one labeled object per image. This mean that sometimes, the models predict a correct object that is not labeled and it doesn't count as a positive match.
- In many scenes, objects are truncated or occluded from other objects.

Both SSD-Lite and Tiy-DSOD reach a speed ($\sim 20FPS$) close to the real-time one. As it is possible to see from the Video Test, it is a sufficient speed to detect objects, if the user is not moving too fast. In the case of SSD-Lite, the results in the nine videos are:

- **slow speed videos**, the model is able to recognize with a good confidence few objects;
- **medium and high speed videos**, the model has difficulties to detect most of the objects, sometimes produce wrong predictions and sometimes right ones;
- **little office videos**, the model begin to recognize, from the door point of view, only some few big objects;

- **medium and big office videos**, the model need to enter in the room to move close to the objects to detect them.

From the videos it is possible to say that, SSD-Lite is able to detect: Desk, Door, Laptop, and Whiteboard, if the user movements are slow and the objects are close enough. Sometime it recognizes also: Bookcase, Keyboard, Monitor, and Window objects. Instead, Tiny-DSOD produces too many wrong predictions with high confidence and this make useless the Video test.



(a) (b)

(c) (d)

Figure 4.7: Prediction examples of SSD-Lite (a)(c) and Tiny-DSOD (b)(d) on OpenImage test set (a)(b) and Office test set (c)(d). A confidence threshold of 50% is used in these predictions.

To overcome some of these weaknesses (mainly the low amount of training images), we proposed another experiments reported in the following section.

4.5.1 SSD-Lite (fine tune)

We may think that an higher amount of object categories could affect the models, bringing to worse results in term of accuracy. That because the model has to learn the representations of more objects. This is true, but not at all. If we compare the detection of a model able to detect only one category with a model able to detect ten different category, the previous statement is right. The real difference between them is that the two models will have networks of different complexity. To detect only one object we can use a simple network, instead, to detect ten different objects we have to use a more complex model. The algorithms we are analyzing has the purpose to detect a indefinite number of object categories, so they are really complex networks. That why an high number of categories doesn't perform worse in respect to a network with less object categories. From this idea and because both the previous models show the need of more training images, we decide to use the Fine Tuning technique. This technique is used to perform a more powerful training, its pipeline is:

1. first the network is trained on a big training set, like the Imagenet dataset, whit all the object categories and all the images.

2. The produced weights are used again as initial value for a new training. This new training is made only with the object categories we need and we can use the dataset we prefer.

3. New final weights are produced and they are able to detect the selected objects.

Thanks to this technique, it is possible to use in the second training a number of images lower then the needed for a normal one (the number depends on the network and on the number of categories). Many famous models provide the so called Pre-trained Weights of their networks, that are trained on famous datasets like: COCO, Pascal VOC, or Imagenet. So, it is easy and faster to re-train the network for the object categories needed. Furthermore, this technique permits to deeply train all the layers of the network. Indeed, as succeeded in our case, many times there are not enough images of the selected objects to train the network from scratch. The fine tuning permits to train all the network layers (before the classification and localization ones) with a big amount of images and later train the last layers (the classification and localization layers) with the images containing only the specific object categories.

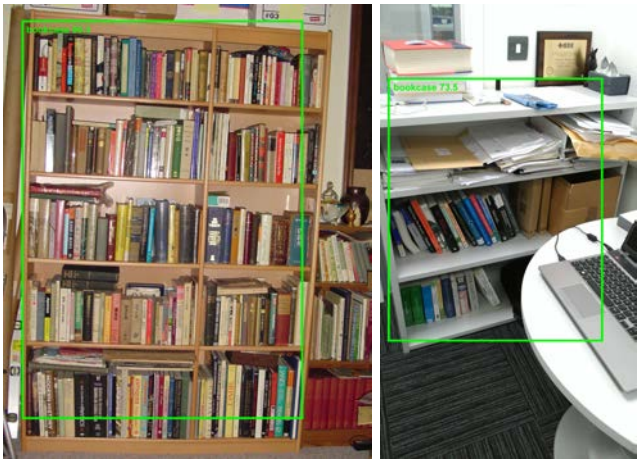
We decided to use the fine training also for our networks. Unfortunately the pre-trained weights of Tiny-DSOD are not provided by the authors, so we are not able to perform this experiment for this network. Instead, it is possible to find the pre-trained weights of SSD-Lite with Mobilenet V2 on [56]. Using the same network setting and taking the weights pre-trained on Imagenet dataset, we perform the training on the same selection of images from OpenImage dataset. The obtained results follow in Tab. 4.4.

As shown in the sixth column of Tab. 4.4 the mean accuracy on OpenImage Test set is doubled. It is not possible to say the same for the mean accuracy of the Office Test set, but the improvements are really evident from the Video test. The new model starts to detect more objects with a good confidence. The predictions are now more frequently correct and with less mistakes (Fig. 4.8a and 4.8b). It is not able to detect all the objects in all the frames, but the improvements are visible. There are again some wrong predictions (few ones) and many missed ones. The model works well when the objects are big in the frame (close to the camera or big objects). Also in the video at medium speed, it detects big and close objects, while in the fast speed videos, it has again difficulties. Again a good confidence threshold results to be 70%.

Obviously, not all the weaknesses of the previous training are now resolved. Indeed, from the Office test set results, it is possible to see that the different distribution of the test images already affects the test, not permitting to reach an higher mean accuracy. In some cases it also perform

Object	SSD-Lite AP		Tiny-DSOD AP		SSD-Lite (ft) AP	
	OpenImage	Office	OpenImage	Office	OpenImage	Office
Backpack	15.5	0.0	3.2	0.0	43.3	14.8
Book	9.1	0.0	0.6	0.0	0.6	0.0
Bookcase	31.4	43.7	14.9	22.6	39.4	9.9
Chair	1.0	0.0	0.1	0.0	1.2	0.0
Desk	11.1	3.1	9.8	5.1	27.9	1.7
Door	17.0	18.2	9.7	8.4	25.0	3.7
Keyboard	18.5	0.2	3.3	0.0	36.5	12.3
Lamp	12.9	0	4.6	0.0	22.82	0.0
Laptop	22.4	19.6	10.0	11.1	43.8	26.1
Light-switch	0.0	0.0	0.0	7.0	77.7	0.0
Monitor	2.4	11.4	5.1	5.6	35.6	4.9
Mouse	15.3	0.0	4.5	0.0	39.7	0.0
Mug	28.8	0.0	5.4	0.0	59.9	0.0
Plant	24.3	60.3	16.4	32.7	24.0	33.3
Telephone	0.0	0.0	0.0	0.0	11.4	0.0
Whiteboard	64.7	7.3	19.3	3.9	68.8	41.1
Window	0.3	0.0	0.8	0.0	2.5	22.2
MEAN	**16.2**	**9.6**	**6.3**	**5.3**	**32.9**	**9.7**

Table 4.4: Results in terms of accuracy of SSD-Lite, Tiny-DSOD, and SSD-Lite fine tuned, each one on OpenImage and Office test set.



(a) (b)

Figure 4.8: Prediction examples of SSD-Lite fine tuned on OpenImage test set (a) and Office test set (b). A confidence threshold of 50% is used in these predictions.

worse then the previous model (Bookcase and Door). It remains also the problem related to little objects and to objects with a low number of images in the Training set. Nonetheless, the network

starts to recognize objects that previously are not detected and in general the predictions are now more homogeneous between a wide range of objects.

In Tab. 4.5 is shown the final performances in terms of accuracy, weight and speed of the three models.

Method	**mAP**		**FPS**	**Params**	**FLOPS**
	OID	**Office**			
SSD-Lite	16.2	9.6	~ 20	4.30M	0.8B
Tiny-DSOD	6.3	5.3	~ 20	0.95M	1.12B
SSD-Lite (fine tuned)	32.9	9.7	~ 20	4.30M	0.8B

Table 4.5: Results in tearms of mean accuracy (mAP), speed (FPS), and weight (Params and FLOPS) of SSd-Lite fine tuned and Tiny-DSOD.

Chapter 5

Conclusions

We have seen in chapter 1 the importance of Object Detection systems in the modern scenario, where they have reached good performances and provide solutions for many issues. They are fundamental and currently used in many research fields and in many work environments. New applications request for light neural networks, that can work on low power devices such as Smartphones and different type of wearables. One example is the Activis Project, that asks for a system able to help visually impaired people to navigate in indoor environments. For this purpose it is fundamental the use of an object detector able to identify the surrounding space. In chapter 3 we propose two possible models able to work on mobile devices and that promise good accuracy:

- SSD-Lite that use Mobilenet V2 [66] as backbone, famous light classification network that is the state-of-the-art for real-time classification on low power devices;
- Tiny-DSOD a novelty proposal by [43] based on DSOD framework, that promises to perform better then SSD-Lite in terms of accuracy and weight.

In chapter 4 we present a new dataset to test the proposed models, called Office dataset. It is made of 459 pictures, captured from a real office room at University of Lincoln, UK. It contains 17 object categories, where each item is captured from different points of view. This dataset provides useful test set for our project and for future researchers that would like to approach it. For the training process we use images coming from the OpenImage V4 dataset [38], that provides a big amount of images and of object categories useful to train the two networks. In addition we created a Video Test set for the same purpose. The video test set does not provide a scientific measure of accuracy, but only a visual feedback, useful to understand how the network is working and to find its weaknesses. The setup chosen includes the use of Tensorflow Lite, Keras and Python 3.6 as code language. The results are computed testing the networks, with an Asus Zenfone AR smartphone, on the Office dataset and on the OpenImage Test set (created from a selection of images of OpenImage V4 dataset). The results highlight the better performances of the SSD-Lite network, but also the lack of images for the training process. Indeed, the complexity of the network doesn't permit to generate good weights, with a low amount of training images, and consequently it doesn't learn well the objects representation. For this reason we train the SSD-Lite with the fine-tuning technique. We use pre-trained weights on Imagenet dataset and we retrain the network using the object categories images. This solution results to be the best one, obtaining an accuracy on the OpenImage Test set of 32.9 mAP and an accuracy on the Office Test set of 9.7 mAP. The network reaches a speed of $\sim$20 FPS, that is enough to perform a real-time detection. The SSD-Lite has many weaknesses, the major given by the training process. It does

not learn to detect all the 17 objects and sometimes provides wrong predictions. Nonetheless, this model constitutes the basis for future updates, new experiments, and new model proposals, that can increase the performances and bring to better solutions in this field.

The SSD-Lite network, produced in this thesis work, is currently used in the development of the Activis project followed by Ing. Jaycee Lock and Professor Nicola Bellotto. This collaboration between the University of Padua, IT, and the University of Lincoln UK, has produces a paper [33] accepted and published at the ICIAP 2019 conference.

Bibliography

[1] Android. Android ndk, 2018. Last accessed 09 September 2019.

[2] Asus. Asus zenfone ar (zs571kl). Last accessed 09 September 2019.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Corinna Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[5] CRCV. Ufc center for research in computer vision. Last accessed 09 September 2019.

[6] Jifeng Dai, Yi Ci Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[7] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:886–893 vol. 1, 2005.

[8] Jun Deng, Wei Dong, Richard Socher, Li-Jia Li, Kuntai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[9] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2155–2162, 2013.

[10] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111:98–136, 2014.

[11] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2009.

[12] Pierluigi Ferrari. Ssd keras. Last accessed 09 September 2019.

[13] Martin A. Fischler and Robert A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, C-22:67–92, 1973.

[14] David A. Forsyth. Object detection with discriminatively trained part-based models. *IEEE Computer*, 32:1627–1645, 2014.

[15] Ido Freeman, Lutz Roese-Koerner, and Anton Kummert. Effnet: An efficient structure for convolutional neural networks. *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 6–10, 2018.

[16] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, volume 5, page 663âĂŞ671, 1995.

[17] Cheng-Yang Fu, Weiwei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. Dssd : Deconvolutional single shot detector. *ArXiv*, abs/1701.06659, 2017.

[18] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[19] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2013.

[20] Ross B. Girshick, Forrest N. Iandola, Trevor Darrell, and Jagannath Malik. Deformable part models are convolutional neural networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 437–446, 2014.

[21] Kristen Grauman and Trevor Darrell. The pyramid match kernel: discriminative classification with sets of image features. *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, 2:1458–1465 Vol. 2, 2005.

[22] Leibe B. Grauman K. *Visual object recognition. Synthesis lectures on artificial intelligence and machine learning*, volume 5. Morgan and Claypool Publishers, 2011.

[23] Bharath Hariharan, Pablo Andrés Arbeláez, Ross B. Girshick, and Jagannath Malik. Object instance segmentation and fine-grained localization using hypercolumns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:627–639, 2017.

[24] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:1904–1916, 2014.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[27] Matthijs Hollemans. Mobilenet version 2, 2018. Last accessed 09 September 2019.

[28] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[29] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.

[30] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara Balan, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2016.

[31] Jonathan Hui. Ssd object detection: Single shot multibox detector for real-time processing, 2018. Last accessed 09 September 2019.

[32] N. Bellotto Jacobus C. Lock, G. Cielniak. Active vision with human-in-the-loop for the visually impaired, 2016.

[33] Stefano Ghidoni Nicola Bellotto Jacobus C. Lock, Andrea Tramontano. Activis: Mobile object detection and active guidance for people with visual impairments. In *ICIAP 2019*, 2019.

[34] Narendra Nath Joshi. The machine learning abstracts (part 1): Classification. Last accessed 09 September 2019.

[35] Achraf Khazri. Face detection models and softwares. Last accessed 09 September 2019.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60:84–90, 2012.

[37] Besir Kurtulmus. Introduction to image saliency detection. Last accessed 09 September 2019.

[38] Alina Kuznetsova. Open images dataset v4, 2018. Last accessed 04 May 2019.

[39] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *CoRR*, abs/1811.00982, 2018.

[40] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2:2169–2178, 2006.

[41] Yali Li, Shengjin Wang, Qi Tian, and Xiaoqing Ding. Feature representation for statistical-learning-based object detection: A review. *Pattern Recognition*, 48:3542–3559, 2015.

[42] Yi Ci Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4438–4446, 2016.

[43] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-dsod: Lightweight object detection for resource-restricted usages. In *BMVC*, 2018.

[44] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head r-cnn: In defense of two-stage object detector. *ArXiv*, abs/1711.07264, 2017.

[45] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. *Proceedings. International Conference on Image Processing*, 1:I–I, 2002.

[46] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2016.

[47] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[48] Lily L. Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *ArXiv*, abs/1809.02165, 2018.

[49] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[50] Jacobus Cornelius Lock, Grzegorz Cielniak, and Nicola Bellotto. A portable navigation system with an adaptive multimodal interface for the blind. In *AAAI Spring Symposia*, 2017.

[51] Jacobus Cornelius Lock, Grzegorz Cielniak, and Nicola Bellotto. Active object search with a mobile device for people with visual impairments. In *VISIGRAPP*, 2019.

[52] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[53] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *ArXiv*, abs/1807.11164, 2018.

[54] Laurence Moroney. Using tensorflow lite on android. Last accessed 09 September 2019.

[55] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. The kitti vision benchmark suite, 2018. Last accessed 09 September 2019.

[56] pkulzc. Tensorflow detection model zoo. Last accessed 09 September 2019.

[57] pkulzc. Tensorflow object detection api. Last accessed 09 September 2019.

[58] Patrick Poirson, Phil Ammirato, Cheng-Yang Fu, Weiwei Liu, Jana Kosecka, and Alexander C. Berg. Fast single shot detection and pose estimation. *2016 Fourth International Conference on 3D Vision (3DV)*, pages 676–684, 2016.

[59] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2015.

[60] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2016.

[61] Joseph Redmon and Ali Farhadi. Yolo: Real-time object detection, 2018. Last accessed 09 September 2019.

[62] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[63] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[64] Olga Russakovsky, Jun Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2014.

[65] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77:157–173, 2005.

[66] Mark B. Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[67] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[68] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1937–1945, 2017.

[69] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[70] Mily Surpless. Actors who are unrecognizable without facial hair. Last accessed 09 September 2019.

[71] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2014.

[72] Tensorflow. Tensorflow lite. Last accessed 09 September 2019.

[73] Sik-Ho Tsang. Review: Xception âĂŤ with depthwise separable convolution, better than inception-v3 (image classification). Last accessed 09 September 2019.

[74] Sik-Ho Tsang. Review: Mobilenetv1 âĂŤ depthwise separable convolution (light weight model), 2018. Last accessed 09 September 2019.

[75] Sik-Ho Tsang. Review: Ssd âĂŤ single shot detector (object detection), 2018. Last accessed 09 September 2019.

[76] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 2013.

[77] Sophia Vassiliadis. Opinion: On autonomous cars, motorcyclists, and culpability. Last accessed 09 September 2019.

[78] VSaaS. Vsaas 2025: Future video surveillance technologies. Last accessed 09 September 2019.

[79] Chi-Feng Wang. A basic introduction to separable convolutions, 2018. Last accessed 09 September 2019.

[80] Amadeus Winarto. Variations of ssd: Deeply supervised object detector (dsod), 2018. Last accessed 09 September 2019.

[81] Wei Xiang, Dong-Qing Zhang, Heather Yu, and Vassilis Athitsos. Context-aware single-shot detector. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1784–1793, 2017.

[82] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2017.

[83] Xin Zhang, Yee-Hong Yang, Zhiguang Han, Hui Wang, and Chao Gao. Object class detection: A survey. *ACM Comput. Surv.*, 46:10:1–10:53, 2013.

[84] Zhong-Qiu Zhao, Peng Zheng, Shou tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2018.

[85] Xizhou Zhu, Jifeng Dai, Xingchi Zhu, Yichen Wei, and Lu Yuan. Towards high performance video object detection for mobiles. *CoRR*, abs/1804.05830, 2018.