Human distance estimation through object detection: a technological implementation

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Davide Brattelli, 921052

Advisor:
Prof. Manuel Roveri

Co-advisors:
Ing. Massimo Pavan

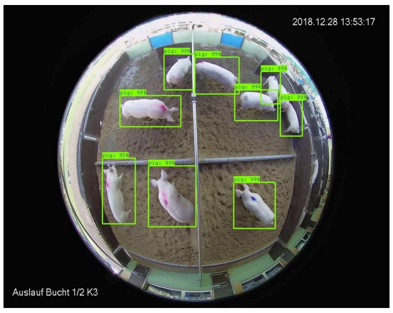
Academic year:
2020-2021

Abstract: Object Detection is one of the major computer vision tasks, that can be applied and incorporated in any field to improve existing techniques or create new ones. The objective of this work is to have an algorithm that is able to detect people and estimate their distance reliably, while also being lightweight and cheap. In this paper, we first give a formal definition of object detection, and describe the incredible evolution it has had in the last decades, focusing on the approaches based on Neural Networks. Then we propose our solution, an application based on the two advanced Object Detection models YoloFastestV2 and SSDLiteMobileNetV3, that can be ran on very limited hardware. We also present a dataset generated by ourselves, that was used to test and evaluate both the detection and the distance estimation.
The results obtained suggest that the environment where the algorithm is applied have a significant impact on its performance, and therefore is a factor to consider when deploying this type of application.

Key-words: Object Detection, AI, TinyML, Computer Vision, Distance estimation

1. Introduction

Object Detection is one of the primary computer vision techniques, which consists in identifying and localizing any entity in an image or video, basically going one step above the classic Image Classification problem where pictures simply get one label. Humans perform this task instantly and continuously, therefore automating it is essential to the simulation of human intelligence in computers and to the automation of other more advanced activities. In the last few years, deep learning techniques have had a significant development; this, together with the overall progress of the whole AI field due to hardware and data growth, has pushed Object Detection onto the spotlight and to frequent breakthroughs. This can be seen in Fig 1 that shows how the number of papers related to this topic has risen throughout the last twenty years.



(a) Animal Tracking. Source: [5]

(b) Object Detection in agriculture. Source: [1]

(c) Vehicle detection. Source: [5]

Figure 2: Object Detection used in different contexts

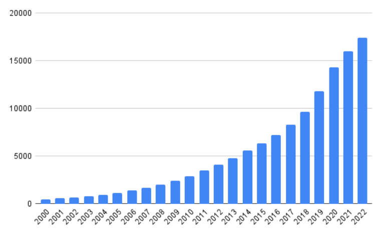


Figure 1: Number of results obtained by searching for "object detection in google scholar

Providing a basic and necessary function, detection algorithms are nowadays the basis upon which more advanced computer vision tasks are built and also the core of most vision-based software and applications. The real-world domains where they can be applied are many and various, such as but not limited to:

- Autonomous Driving
- Security
- Medical diagnostics
- Agriculture

Some examples of Object detection applied to very diverse fields can be seen in Fig 2.

One of the challenges that arise in the deployment of Object Detection algorithms in real-life is the trade-off between accuracy and run-time. It is easy to see how many applications require that the objects are identified almost instantly; for example, in a self-driving car a very accurate but slow detection would be worthless and bring the whole system to failure. This is a critical issue, since object detection models are extremely large and heavy, and need a great deal of time and computing power to be executed. Moreover this obstacle becomes even harder when taking into account the constraints on hardware as well; for instance, a Robot would have an Object detection agent that must share limited resources with all the other programs that are running simultaneously, and are equally important like motion or audio recognition. Another world that particularly exploits this type of image analysis is IoT; here size, energy consumption and cost become very determining factors in the success of a product.
It follows that working on the accuracy-speed trade-off and making object detection affordable on lower-end devices is vital to the advancement of object detection techniques and their use in real-life scenarios. This is the point this thesis focuses on, by looking into and implementing some of the most recent and state-of-the-art object detection algorithms, and evaluating their performances in a practical case. Firstly, we will give a generic introduction to Object Detection and the different approaches that can be used to achieve it; then we will see the mechanisms behind neural networks and deep learning; lastly we will evaluate the performance of some models in real-life experiments in a closed environment.

2. Related Literature

Object Detection is a task that has been central in researches in the area of Computer Vision, which is defined as the AI field which gives to computers the ability of understanding meaningful information from images, videos, or any other visual inputs [2].

The field of computer vision encompasses many visual recognition problems, not limited to just Object Detection. In [54], these four are the main ones considered:

- Image Classification: the goal is to recognize the semantic categories of entities in the given image.

  INPUT = Image containing a single object.
  OUTPUT = One or more labels, each with a corresponding score.

- Object Detection: the goal is to not only recognize the categories, but also predict their locations in the image.

  INPUT = Image containing one or many objects.
  OUTPUT = One or more bounding boxes,i.e. boxes surrounding the objects, each having a class label and a score.

- Semantic Segmentation: pixel-level classifier, which provides higher detail and therefore a better understanding of the picture, but cannot differentiate different objects belonging to the same category [28];

  INPUT = Image containing one or many objects.
  OUTPUT = Full Resolution image, where each pixel is associated with a class label.

- Instance Segmentation: fusion of object detection and semantic segmentation, obtaining pixel-level localization.

  INPUT = Image containing one or many objects.
  OUTPUT = Full Resolution image, where each pixel is associated with a class label and the specific instance it belongs to.

The above-mentioned tasks and their typical representation when applied to the same image are shown in Fig 3.
Each technique has its own advantages/disadvantages and must be used in the proper problem. For example, instance segmentation is often used in the medical field where the pixel accuracy is very valuable, while the higher computational cost that comes with it is not an issue; examples are [21] and [51]. In our case, recognizing and locating the patients in their room, and possibly understanding their activity, does not require the accuracy that segmentation offers; moreover, keeping the complexity, and consequently the cost of the system, as low as possible is fundamental. Due to these factors, we consider the Object Detection framework to be the most suitable one.

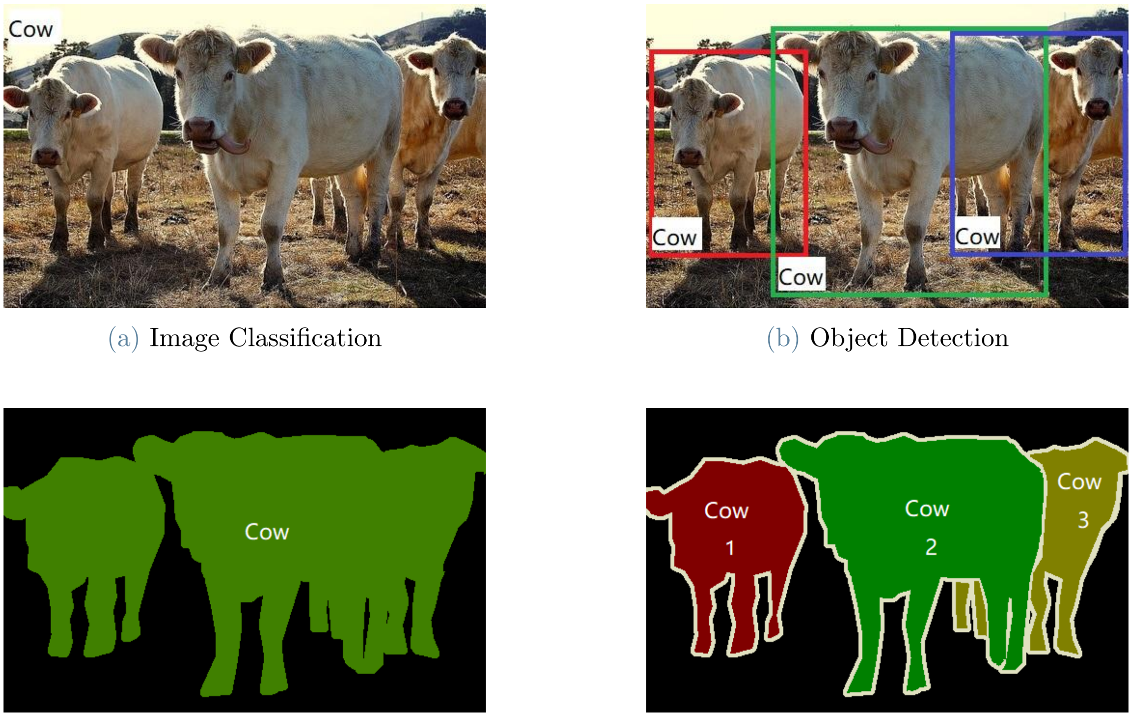
2.1. Definition

A formal definition of Object Detection is given by Vershae & Ruiz-del-Solar [3]:

> Given a set of object classes, object detection consists in determining the location and scale of all object instances, if any, that are present in an image. Thus, the objective of an object detector is to find all object instances of one or more given object classes regardless of scale, location, pose, view with respect to the camera, partial occlusions, and illumination conditions.

2.2. Problem setting

The purpose of this section is to formally define the object detection problem; an example of this is presented in [54]. Object detection can be seen as composed of two subtasks: object classification and object localization. The detector must be able to distinguish objects from the background, assign them to their corresponding class label, and localize them by predicting a bounding box.
A training set made up of N images $\{x_1, x_2, ..., x_N\}$ that are annotated, i.e. each image x_i comes with a set of



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

Figure 3: Results obtained by applying the four methods on the same image. Source: [54]

known objects M_i with their corresponding class and location in x_i:

$$y_i = \{(c_1^i, b_1^i), (c_2^i, b_2^i), ..., (c_M^i, b_M^i)\} \tag{1}$$

where c_j^i and b_j^i respectively represent the class label and the bounding box of the j^{th} object in the i^{th} image. The prediction algorithm is denoted as f with parameter vector θ, while the prediction has the same shape as the training set:

$$y_{pred}^i = \{(c_{pred_1}^i, b_{pred_1}^i), (c_{pred_2}^i, b_{pred_2}^i), ..., (c_{pred_M}^i, b_{pred_M}^i)\} \tag{2}$$

To optimize the predictor, we utilize a loss function defined as such:

$$l(x, \theta) = \frac{1}{N} \sum_{i=1}^{N} l(y_{pred}^i, x_i, y_i; \theta) + \frac{\lambda}{2} \|\theta\|^2 \tag{3}$$

where the second term is a regularizer, with hyperparameter λ, that penalizes high parameter values, and thus lowers model complexity and overfitting.

To evaluate the quality of the predicted location, therefore compare the known bounding box b_{gt} with the computed one b_{pred}, a metric called Intersection-over-Union (IoU) has been defined:

$$IoU(b_{pred}, b_{gt}) = \frac{Area(b_{pred} \cap b_{gt})}{Area(b_{pred} \cup b_{gt})} \tag{4}$$

An example of IoU computed on a detection is shown in fig 4. To state whether or not the prediction *tightly* covers the object, an IoU threshold Ω is used; this is commonly given the value $\Omega = 0.5$.

A prediction is considered positive if the class label is correct and the localization satisfies the IoU threshold:

$$Prediction = \begin{cases} Positive & \text{if } c_{pred} = c_{gt} \text{ and } IoU(b_{pred}, b_{gt}) > \Omega \\ Negative & \text{otherwise} \end{cases} \tag{5}$$

Generally, to evaluate and compare different object detection algorithms, Mean Average Precision (mAP) over C classes is used. Along with detection accuracy, the inference speed of the models must also be taken into account when evaluating them; this is done by considering how many Frames per Second(FPS) it is able to work at. A detector is considered real-time if it achieves at least 20 FPS.

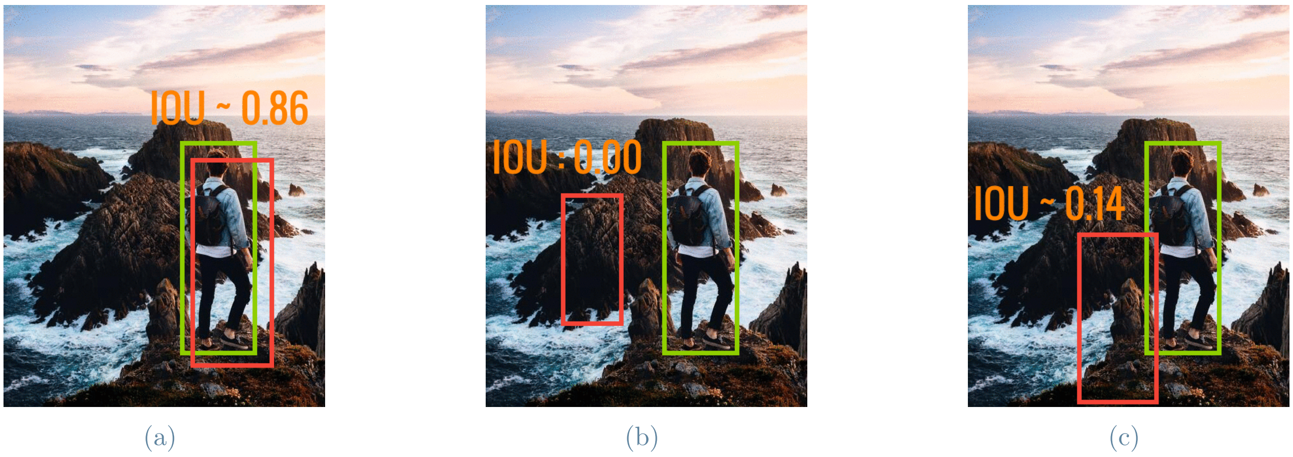


Figure 4: IoU example on human detection. Source: [25]

2.3. Deep Learning Notions

As we have seen in the last chapter, basically all modern object detection approaches consist of Deep Neural Networks (DNN). Therefore in this section we will introduce the concept of Neural Networks (NN), their mechanism and some important architectural paradigms.

Artificial Neural Networks (ANN) are mathematical models inspired by the way the human brain work and indeed the building block of any NN of a neuron: receives an input from another neuron, performs some kind of computation, and outputs something. This structure is shown in Fig 5. A formal mathematical definition of a neuron output is:

$$y = \phi(x) = h(w^T x + b) \tag{6}$$

where w,x$\in R^N$ respectively represent the weight vector and the input vector, N is the input size, and b is a real-value called bias. The real value obtained by the sum is then fed into a non-linear function h, called activation function; there are many types of activation functions, and we'll show them later.

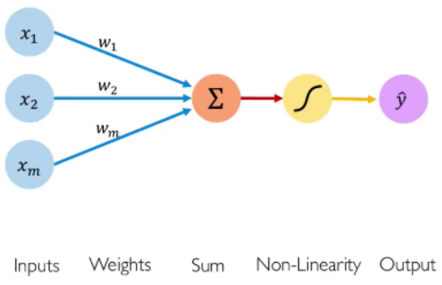


Figure 5: Structure of a neuron.

Neural Networks are obtained by connecting the neurons in some kind of way, and the neurons located at the same depth are grouped a so-called layer. Generally, the depth of a NN denotes the number of non-linear transformations between the layers, and the width instad is given by the dimensionality of a layer, i.e. the number of neurons it contains [10]. Usually, for an architecture to be considered deep it needs to have at least 2 hidden layers; it follows that Deep Learning is simply the machine learning branch which utilizes that type of NNs, and that is the context of our work as well.
There are numerous types of network architectures used in deep learning; here we simply give a brief overview of the most important and basic ones, before going into detail in the ones mostly used in image processing.

2.3.1 Feed Forward Neural Networks (FFNNs)

FFNNs are the networks that do not present any cycles, consequently the data goes from input and output without circling back and therefore no "memory" is maintained. This type of network basically encodes a set of numbers taken as input into a prediction, that can either be discrete (classification) or continuous (regression). Fig 6 contains a scheme illustrating FFNN mechanism.
The output of a FFNN is defined as follows:

$$y_l(x,w) = \sum_{j=0}^{m} w_{lj}^{(2)} \phi_j(x) = \sum_{j=0}^{m} w_{lj}^{(2)} h(\sum_{i=0}^{d} w_{ji}^{(1)} x^i) \tag{7}$$

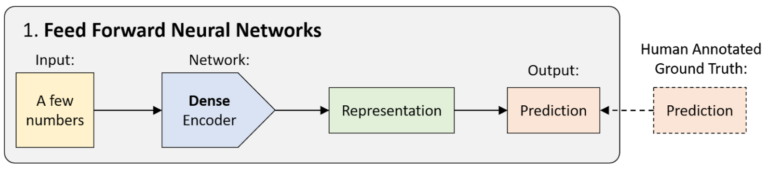


Figure 6: Structure of a FFNN. Source:[15]

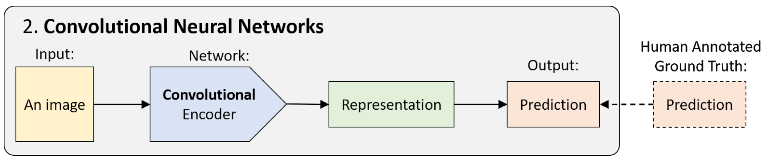


Figure 7: Structure of a CNN. Source: [15]

where $\phi_0(x) = 1$, $w_{lj}^{(2)}$ and $w_{l0}^{(2)}$ are the weights and the biases on the arcs between the hidden layer and the l^{th} node of the output layer, $w_{ji}^{(1)}$ and $w_{j0}^{(1)}$ are the weights and biases on the arcs between the input and the $j^{(th)}$ node of the hidden layer. The weights $w^{(1)}$ and $w^{(2)}$ are concatenated into the parametere vector θ.
During training, iterative optimization methods are used to update Φ in order to minimize some kind of error function $E(\theta)$:

$$\theta^{(\tau+1)} = \theta^{(\tau)} + \Delta\theta^{(\tau)} \tag{8}$$

$\Delta\theta^{(\tau)}$ is commonly computed with respect to the gradient of the error function $\nabla E(\theta)$, for instance with gradient descent.

2.3.2 Convolutional Neural Networks (CNNs)

Differently from traditional ANNs, where each neuron is a layer is connected to all neurons in the next one which causes the presence of a huge number of parameters, in CNNs a neuron is only connected to nearby neurons in the next layer. This is done because CNNs exploit the space-invariance trick to efficiently learn local patterns, most commonly in images. The space-invariance trick implies that a pattern, e.g. a human face, has the same features no matter where it is located in the image. Fig 7 shows the structure of CNNs.

The building blocks of a CNNs are the following layers: input layer, activation layer, convolutional layer, pooling layer, and fully connected layer [39]. ReLu and pooling layers perform fixed operations (no parameters), while the convolutional one and the fully connected one have parameters that are trained to minimize the error function.

Convolutional Layer

Core layer of CNNs, which means they make up most of the computation in the network. A convolutional layer slides at least one kernel, basically a grid of weights, across the input and at each step performs a convolution between the input and the kernel, and the results are stored in an activation map. This process is regulated by 3 hyperparameters: kernel size, stride (how much the kernel should move at each step), and zero-padding. Additionally there are many types of convolution operations: ordinary convolution, transposed convolution, hole convolution, and depth separable convolution. We are interested in Depth Separable Convolution since it is used in lightweight network MobileNet, which is the backbone in one of the proposed solutions; in this type of convolution, a filter is applied to each input channel by depthwise convolution, then the outputs obtained this way are fused with pointwise convolution. The achieved decomposition greatly reduces the amount of computation and the size of the model, that is a great plus in object detection, especially in embedded devices.

Activation Layer

This layer simply applies an activation function, so that the network can model non-linear relationships. Common activation functions are: Rectified Linear Unit (ReLU), Randomized Leaky ReLU (RReLU), and Exponential Linear Unit. The ReLU one is simply:

$$f(x) = max(0, x) \tag{9}$$

Pooling Layer

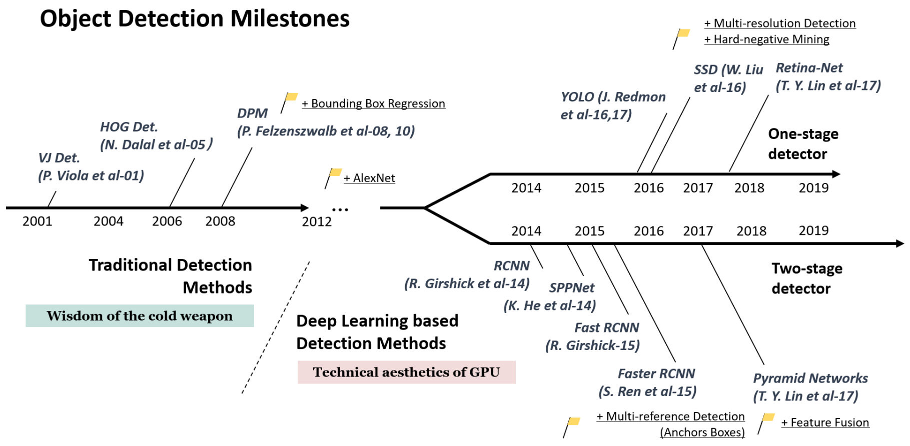

Object Detection Milestones
VJ Det. (P. Viola et al-01)
HOG Det. (N. Dalal et al-05)
DPM (P. Felzenszwalb et al-08, 10)
+ Bounding Box Regression
+ AlexNet
2001
2004
2006
2008
2012
Traditional Detection Methods
Wisdom of the cold weapon
Deep Learning based Detection Methods
Technical aesthetics of GPU
+ Multi-resolution Detection
+ Hard-negative Mining
YOLO (J. Redmon et al-16,17)
SSD (W. Liu et al-16)
Retina-Net (T. Y. Lin et al-17)
One-stage detector
2014
2015
2016
2017
2018
2019
Two-stage detector
RCNN (R. Girshick et al-14)
SPPNet (K. He et al-14)
Fast RCNN (R. Girshick-15)
Faster RCNN (S. Ren et al-15)
Pyramid Networks (T. Y. Lin et al-17)
+ Multi-reference Detection (Anchors Boxes)
+ Feature Fusion

Figure 8: Timeline of Object Detection technology. Source: [56]

Usually located right after a convolutional layer, pooling layers' purpose is to reduce the dimension of the input while conserving as much information as possible, by applying some kind of pooling technique. According to Scherer et al. [43], pooling also reduces overfitting and improves the generalization ability of the network. The well-known pooling methods are: averaging-pooling, min-pooling and some advanced pooling methods, like fractional max-pooling and stochastic pooling. The most common one is max-pooling, because it is better at capturing invariances [43].

Fully connected layer
In CNNs, one fully connected layer is usually placed right before the output layer; however this practice has been criticized for the number of parameters it adds [44].

2.4. Typical approaches

Object Detection is a technology that has been around since the 1960s, with applications like character pattern recognition and assembly and verification processes [27].In the last few years, technology has had an incredible evolution: proliferation of high-end computers and availability of high-quality cameras and images. These aspects make Object Detection an attractive topic and have made this task develop tremendously as well. As a consequence, numerous different methods have been employed to solve the need for accurate and fast detection, and each year there are new breakthroughs. This makes keeping up with the advancements quite hard and confusing. For this reason, in this section we will briefly give an overview of the main steps the field has gone through, in order to have a clear understanding of the path that Object Detection has had, and where it is heading at this very moment as well. Zou et al. [56] divide the "life" of Object Detection into two time periods: Traditional Object Detection period before 2014, and Deep Learning Based Detection period after 2014. Fig 8 shows the timeline of Object Detection and its development since the early 2000s.

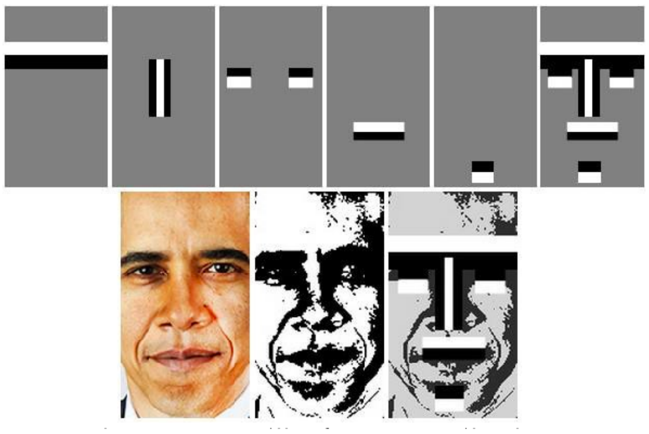


Figure 9: Haar-like features application. Source: [23]

2.4.1 Traditional Detection Methods

Because of the lack of effective image representation that we have nowadays, the first detection algorithms were built based on handcrafted features: depending on what the model needed to recognize, complex features representations had to be designed; moreover, to make up for the low computational power, some techniques to speed up the algorithms were used [56].

Viola Jones Detector
In 2001, the Viola-Jones Detector was developed to face the face detection problem, achieving for the first time real-time detection of human faces without auxiliary information like pixel color or image differences [49]; the model could run at 15 FPS on a 700MHz Intel Pentium III. This technique consisted in using sliding windows and Haar basis functions rather than pixels. The Haar-like features obtained are black and white rectangular boxes combined in a specific way as to resemble traits of human faces, e.g. black box for the eyes, white box for the upper cheeks; an example of Haar-like features is shown in Fig 9.
Three main features made the Viola-Jones detector distinguished it from its peers:

- Integral image, an intermediate representation that makes the computational complexity of each sliding window independent of its size;
- Feature selection, the boosting meta-algorithm AdaBoost was used to select a small set of critical features;
- Attentional Cascade, a multi-stage detection process much like a decision tree, which increases the speed of the detector by filtering out unimportant parts of the image and focusing on the promising ones.

HOG Detector
Mostly motivated by pedestrian detection, the Histogram of Oriented Gradients descriptor was proposed in 2005 by N. Dalal and B. Triggs and is based on the idea that an object's shape can be obtained by the gradient vectors [7] . The HOG descriptor uses local contrast normalization and divides the input image into a grid of cells; then for each cell it creates a histogram of gradient directions or edge orientations over the pixels contained in the cell. The combined features produced this way, called HOG features, are then fed into a SVM classifier. To deal with different sized objects, HOG keeps the detection window constant while changing the scale of the input image.
Fig 6 shows how the gradient histogram graph actually gives a good representation of the two people in the initial image.

Deformable Part-based Model
Extension of the HOG detector proposed by Felzenszwalb in 2008 [12] and then further improved by Girshick [13, 14]. It follows a "divide and conquer" strategy, where the training becomes learning how to decompose an object, and the inference becomes the detection of the different parts of the object. A DPM detector is composed of a root-filter and some part-filters. The configurations for the latter are not manually set, but instead a weakly supervised learning algorithm is used to automatically learn them.
Girshick reformulated this process in the shape of Multi-Instance Learning and applied new techniques such as "hard negative mining", "bounding box regression", and "context priming", that are still relevant today.

2.4.2 Deep learning based detection methods

Object detection with handcrafted features reached a plateau at the start of the 2010s. On the other hand, Deep Learning came back into the spotlight in 2012 with AlexNet winning the ImageNet Large Scale Visual

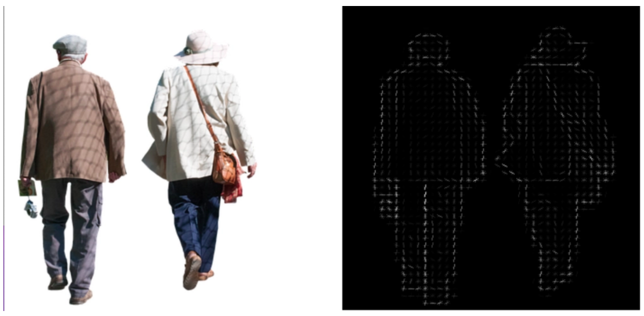


Figure 10: Initial image and intermediate HOG features representation. Source: [27]

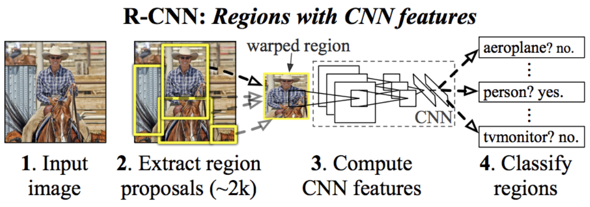


Figure 11: Scheme showing the overall structure of R-CNN. Source [17]

Recognition Challenge (ILSVRC)[26].
Deep Learning approaches can be divided into two main categories: two-stage detectors and one-stage detectors.

Two-stage detectors perform two different tasks separately:

1. Proposal generation, portions of the image which contain objects are identified;
2. Classification, the regions obtained in the first stage are fed into a model that assigns them a class label;

R-CNN

Region-Based Convolutional Neural Network (R-CNN) was proposed in 2014 by Girshick et al. [17]. The system is comprised of three modules: region proposal, feature extraction, and classification. Candidate object boxes (around 2000 per image) are generated through selective search [48]; each proposal is then brought into a fixed scale image and fed into a ImageNet trained model to extract a 4096-dimensional feature vector; in the last step, class-specific linear SVM classifiers are utilized to determine if the proposal contain an object and which class it belongs to; the bounding boxes for the scored regions are refined using a bounding box regression.
Although R-CNN is very accurate, and performed better than the traditional detection methods, it had some severe flaws which made it so it could not be implemented in real-time: heavy duplicated computations for the different proposals, the three modules being indendent and not optimizable end-to-end, selective search relying on low-level details.

SPPNet

Introduced by He et al [18] in 2014, The SPPNet architecture improved on the R-CNN one by using the theory of Spatial Pyramid Matching (SPM) and adding a Spatial Pyramid Pooling (SPP) layer, which allows for the extraction of fixed-length feature vectors, independently of the size of the image and without rescaling it; the result is a network that is faster than R-CNN, can learn more discriminative features, and also does not suffer from information loss or geometric distortion due to not needing to modify the image. Nonetheless, SPPNet is still afflicted by the other R-CNN flaws: multi-staged training, only fine-tuning its fully connected layers.
Fig 6 illustrates the SPP layer's mechanism.

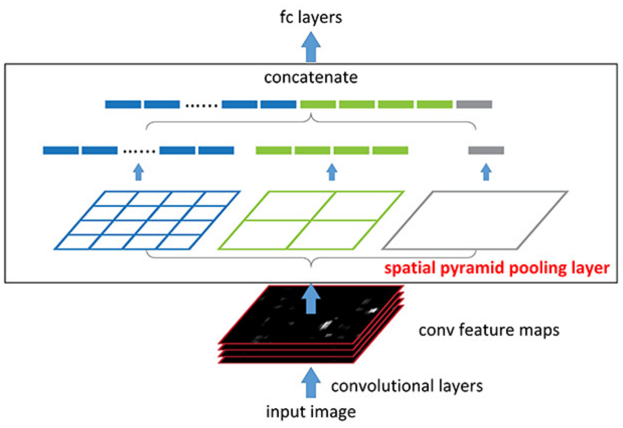


Figure 12: Diagram of the mechanism implemented in a SPP layer. Source: [18]

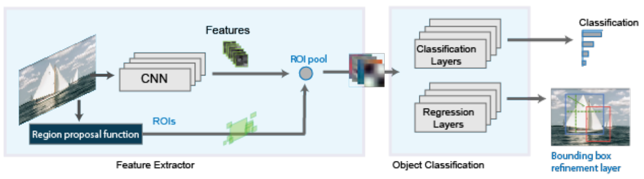


Figure 13: Fast R-CNN architecture. Source: [6]

Fast R-CNN

The following year, Girshick addressed the issues affecting R-CNN and SPPNet with Fast R-CNN [16], a multi-task detector whose main new feature was a RoI pooling layer, a special case of SPP layer with only one pyramid level. Like in SPPNet, the whole image is processed with convolutional layers to produce feature maps; then feature vector from each proposal region is extracted by this layer, goes through a sequence of Fully Connected (FC) layers, and in the end splits into two output layers: classification and regression. The former produces softmax probabilities for the $C+1$ categories, while the latter predicts the bounding boxes encoded as four real values. The Fast R-CNN architecture is shown in fig 8.

Unlike the previously examinated architectures, all three phases in R-CNN can be trained end-to-end. In order to achieve this, the following multi-task loss function is defined:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{10}$$

where $L_{cls}(p, u) = -\log p_u$ represents the log loss for ground truth class u and p_u is the u^{th} element of the discrete probability distribution $p = (p_0, p_1, ..., p_C)$ over the $C + 1$ outputs of the last FC layer. $L_{loc}(t^u, v)$ is the loss between the predicted boxes $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$ and the regression targets $v = (v_x, v_y, v_w, v_h)$.
Despite being a clear improvement both in speed and accuracy, Fast R-CNN's detection speed remains limited by the region proposal mechanism.

Faster R-CNN

The issue still holding object detection back was the reliance on additional methods to generate candidate region proposals; these methods, one example of them being Selective Search, could not be learned in a data-driven way. Ren et al. fixed this with the development of Faster R-CNN [40], which introduced the new Region Proposal Network(RPN): a separate network responsible with the proposal generation that could be trained in a supervised manner. RPN consists of a fully-convolutional network, that is capable of predicting bounds and scores at each position simultaneously; this is done by using a sliding window over the feature map, obtaining a feature vector for each position, that is fed into two siblings FC layers, one for classification and one for regression. The results next go through the final layer for the actual object classification and localization. Since the RPN works at a specific convolutional layer, it shares the previous layers with the object detection network, making it almost cost-free. The architecture of Faster R-CNN can be found in fig 9.

The regression towards true bounding boxes is done by comparing proposals relative to reference boxes, called

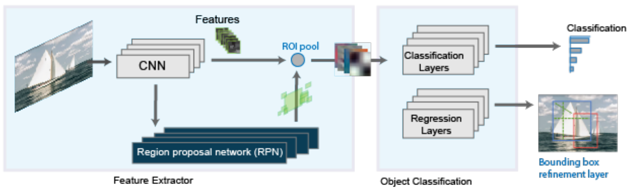


Figure 14: Faster R-CNN architecture. Source: [6]

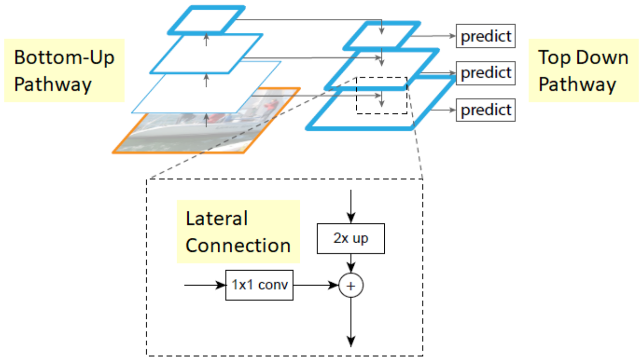


Figure 15: Structure of FPN. Source: [47]

anchors. R-CNN adopts anchors with 3 different scales and 3 different aspect ratios. The loss function is:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{11}$$

where p_i represents the computed probability that the i^th anchor contains an object, while p_i^* is the ground truth label and assumes values:

$$p_i^* = \begin{cases} 1 & \text{if the anchor is positive} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

On the other hand, t_i is the vector of four real values that encodes the predicted bounding box, and t_i^* is the ground truth box.

FPN

A problem that afflicted Faster R-CNN is that, like its predecessors, it only used a single layer feature map to make its prediction, leading to trouble when dealing with small entities. In 2016, T.-Y. Lin et al. [30] proposed Feature Pyramid Networks, based on the fact that in Deep Convolutional Neural Networks (DCNN) the features at deeper layers are semantically strong, thus leading to good classification, but spatially weak, so not ideal for the localization stage. FPN combines the spatial information from shallow layers with the semantic one coming from deeper layers; this is achieved by adding to the usual bottom-up pathway a top-down pathway and several lateral connections. The described structure can be seen in Fig 12.
The bottom-up pathway is the classic feedforward backbone ConvNet, which produces a pyramid of feature maps by iteratively downsampling the maps with a stride of 2. The output of the last layer of each stage is used as the reference set of feature maps to enrich the top-down pathway via the lateral connections.
The top-down pathway consists of higher resolution feature maps getting unsampled by a factor of 2 using the nearest neighbour, and then enhanced through being merged with those of the same size from the bottom-up pathway.

Contrasting two-stage detectors, one-stage detectors get rid of the bottleneck caused by handling the different stages, and perform global regression straight from image pixels to bounding boxes and classes; this results in higher detection speed, but at the expense of accuracy. We will briefly give an overview of the evolution of this framework, before diving into two of the most important one-stage architectures which are the backbone of our application: YOLO and SSD.

Prior to the development of YOLO, there were various attempts at creating a detector as a one shot regressor. We take a look at some of them: Szegedy et al. [45] proposed a DNN-based regression towards an object mask, which however had issues dealing with overlapping objects. Pinheiro et al. [35] with a network that after the initial feature extraction split into two branches, one predicts a segmentation mask for the object while the other predicts an object score. Erhan et al. [11] with a network called "DeepMultiBox", which achieves class-agnostic object detection and predicts a fixed number of bounding boxes which represent possible objects. AttentionNet [55] proposed by Yoo et al. generates from the top-left and bottom-right corner of an image quantized directions pointing at a target, which are then used to recursively crop the image until it converges to a bounding box fitting the target object; it works exactly like an ensemble learning method, that sequentially trains weak learners one on top of the other to produce a strong learner; The mechanism of this peculiar network is shown in Fig 13. Najibi et al. [33] with the network G-CNN based on training a convolutional network to scale and move the boxes towards objects; During the training phase, each object box is assigned to a ground truth object by a function that considers the IoU, then at each step regression is used to move the boxes towards the object assigned to them; during the test phase, for every box at every iteration, scores are computed and the box is adjusted by the regressor for the most probable class. This approach has difficulties when faced with small or highly overlapping objects. Some examples of the above-mentioned models are shown in Fig 13.

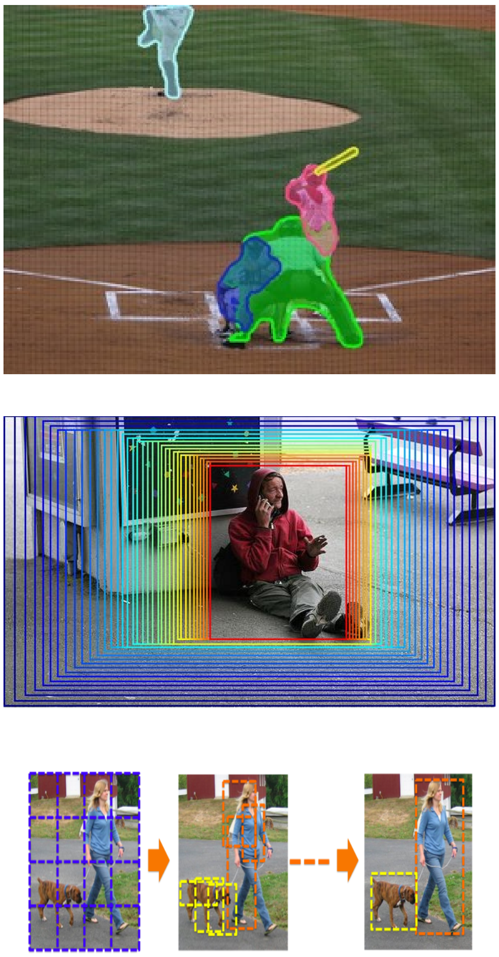


Figure 16: Starting from the top: (a) Szegedy's approach to obtain object masks (source:[45], (b) AttentionNet recursive image cropping (source:[55], (c) G-CNN boxes being trained (source:[33])

YOLO

YOLO, which stands for You Only Look Once, was proposed in 2015 by R. Joseph et al. [36] is one of the major one-stage detectors. The idea behind it is to divide the input image in an SxS grid and each cell is responsible for predicting the class and the bounding box of any object centered in it. YOLO was extremely fast: it could reach 45 FPS, and even 125 FPS with a simpler backbone. Nonetheless, it suffered from two main cons: being able to detect only up to two objects per location, making small and/or crowded entities hard to deal with, and using only the topmost layer for the prediction, causing problems for objects at different scales.
After its first version, many improved versions of YOLO have been created and is still a state-of-the-art algorithm. Further analysis of YOLO is provided in the next section.

SSD

Published in 2016 by Liu et al. [32], Single-Shot Multibox Detector fixed the limitations of YOLO. Similarly to YOLO, SSD divided the image into grid cells, but in addition it used a set of anchor boxes to discretize the output space of bounding boxes. Moreover, to detect objects at different scales, it predicted on different feature maps at different heights of the architecture and obtained the final prediction by merging the multiple predictions. SSD achieved similar accuracy to Faster R-CNN, but also had the advantage of being real-time. Further analysis of SSD is provided in the next section.

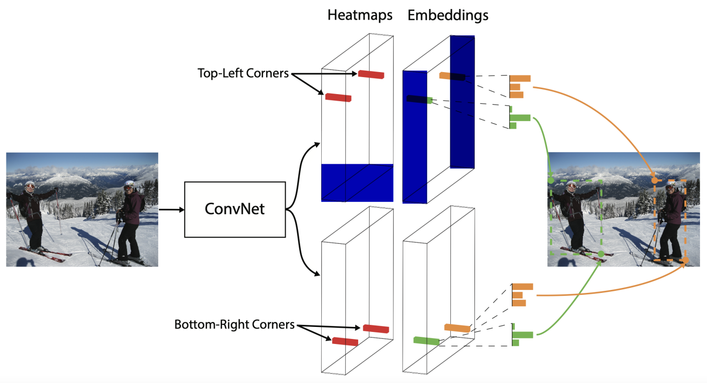


Figure 18: Basic structure of the CornerNet network. Source: [29]

RetinaNet

A drawback of one-stage detectors compared to two-stage ones was not being able to filter out easy negative samples due to the absence of a proposal generator, which resulted in heavy class unbalance, with thousands of candidate proposals per image but only a few actually containing objects. In practice this meant: inefficient training, easy negatives could overwhelm training and bring degenerate models. This issue was fixed in 2017 by Lin et al. with RetinaNet [31], a one-stage detector which employed focal loss to handle the class balance.

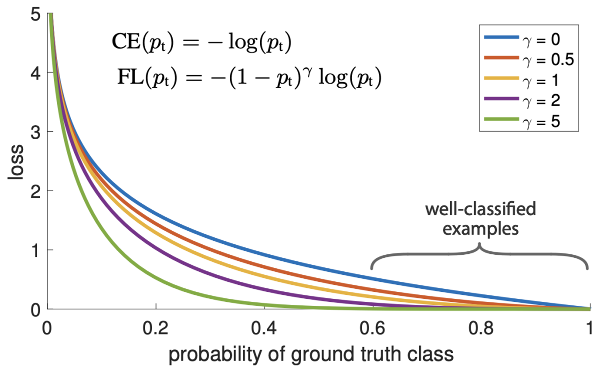


Figure 17: Trend of the focal loss with different values of the hyperparameter γ. Source: [31]

The cross-entropy (CE) loss for binary classification is defined:

$$\mathrm{CE}(p, y) = \begin{cases} -\log p & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise} \end{cases} \tag{13}$$

where $y = \pm 1$ is the ground truth label and $p \in (0, 1)$ represents the predicted probability of belonging to the class corresponding to value 1. For convenience define p_t:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases} \tag{14}$$

The focal loss is:

$$FL(p_t) = -(1-p_t)^\gamma \log(p_t) \tag{15}$$

As can be seen in Fig 17, the hyperparameter λ determines the rate at which easy samples are downweighted, and when $\lambda = 0$ Focal Loss reduces itself to CE. This approach reduces the contributions to the loss by easy samples and extends the range where low loss is attributed, therefore compensating for the class unbalance.

CornerNet

Unlike all the previous approaches, Law and Deng proposed CornerNet [29], an anchor-free model to get rid of the complexities that anchor-based models have: necessity of huge anchor sets, and the many hyperparameters and design choices needed. In CornerNet, objects are detected as a pair of keypoints, the top-left and bottom-right corners of its bounding box. A single convolutional network two heatmaps (one for TL corners, one for BR corners) of size HxW and depth C (number of categories) that computed the probabilities of being corners, and an embedding vector for each corner which has the function of pairing corners belonging to the same object. Fig 18 shows a simplified scheme of the CornerNet architecture.

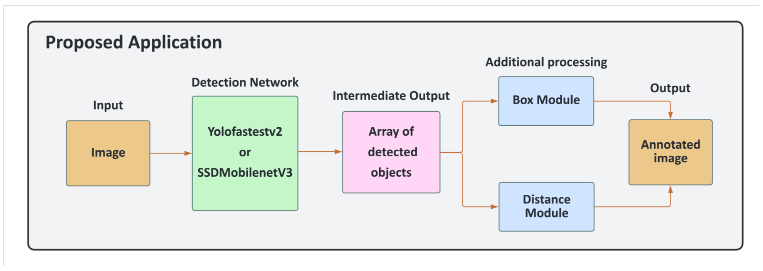


Figure 19: Architecture of our solution.

3. Solution description

The purpose of our work is to obtain an application that is suitable for the detection of patients in hospital environments, and additionally is able to estimate the distance of said patient from the camera itself and can be ran on an embedded device. In order to not put any patient in danger, the detection algorithm must be real-time, and sufficiently accurate, while also being easily deployable in many rooms, possibly even with multiply angles of the same room; for this reason we chose to adopt two of the most recent and advanced lightweight detection models:

- YoloFastestv2;
- SSDLiteMobileNetV3;

3.1. Application architecture

The application we have implemented can be seen, in a high-level view, as composed by 3 computational modules: Detection, Box processing, Distance processing; and by 3 data representations: Input, Intermediate Output, Output. This structure can be seen in the diagram in Fig 19.
Next we give a more detailed decription of the components present in the architecture:

- Input: this module is responsible for obtaining input images from any source; in our case it can be a camera connected to the device or from files. For simplicity, the image then gets resized to a 320x320 resolution and fed into the detection network.
- Detection: module based either on YoloFastestV2 or SSDLiteMobileNetV3, which takes an image and outputs an array containing every object detected in the input image; the objects are connotated, which means that each entry is composed of: category of the object, assigned score, and 4 real-values (x, y, w, h), respectively x and y of the bottom-left corner, height and weight of the bounding box. This data representation is then passed on to the Box module and the Distance module.
- Box module: deals with overlapping bounding boxes that were assigned to the same category, to avoid multiple boxes for the same object; obviously this leads to issues when dealing with crowded objects. Finally, it draws the boxes on the output image.
- Distance module: given the height of the person, and camera hardware specifications, this module computes an approximation of the distance from the camera at which any object assigned to the category "person" stands. The mechanism behind this module will be further explained later.

The pseudo-algorithm for the application is shown in 1. In practice, we developed two distinct programs, one with YoloFastestV2 and one with SSDLiteMobileNetV3; both of them can be found at the following links: Yolo-version, SSD-version.

Algorithm 1 Name of the Algorithm

```
    Input: person_height, camera_specs, detection_threshold, mode
 1: if $ifmode == 0$ then
 2:     input = initiateCamera();
 3: else
        input = openFile();
 4: end if
 5: while image_available do
 6:     image = input.takeFrame();
 7:     resize(image, 320);
 8:     object_array = detection(image, detection_threshold);
 9:     for entry in object_array do
10:         drawBox(entry);
11:         if entry.cat=="Person" then
12:             computeDistance(entry);
13:         end if
14:     end for
15: end while
```

3.2. YOLO architecture overview

As we have seen in Section 2, YOLO is a major one-stage detector that has been around for quite a few years, during which many updated versions of it have been released; thanks to these, the YOLO paradigm is still one of the top detection algorithms. Here we will describe the evolution of YOLO through its different versions until the one that we adopted, called YoloFastestV2.

3.2.1 YOLOv1

YOLO, like other one-stage detectors, does not perform a proposal generation step, but just predicts directly from the image. The first version YOLOv1 has the following workflow:

1. Train a CNN to perform image classification;
2. Take the input image and split it into a SxS grid. Each cell is responsible for the objects whose center falls inside of them, and must detect them. What detection means here is simply predicting 3 things: B bounding boxes, confidence score, probabilities to belonging to each class.
   A bounding box is defined by a tuple of 4 values $< center_x, center_y, width, heigth >$; these are normalized with respect to the image size, so they all fall in the interval $[0, 1]$.
   A confidence score expresses the probability of the cell actually contains an object. It is formally defined as:
   $$Conf = Pr(object) * IoU(pred, truth) \tag{16}$$
   If no object exists in the cell, $Conf$ will have value 0.
   C (number of categories) class conditional probabilities,that are conditioned on the presence of the object in the cell itself, are computed for each cell.
   $$Pr(\text{object belongs to } Class_i|\text{Object is in the cell}), \text{ for } i = 1...C \tag{17}$$
   What this all means is that each image contains $SxSxB$ bounding boxes; since every box is encoded with 4 values, confidence, and class probability, for each image YOLO predicts $S \times S \times (5B + C)$.
3. The final layer of the trained CNN is adjusted so that it outputs a tensor of size $S \times S \times (5B + C)$.

The process above explained is shown in Fig 20.

Network architecture

The network architecture used in YOLOv1 is inspired by the GoogleNet model for image classification: it contains 24 convolutional layers followed by 2 fully-connected layers, and replaced the inception modules used in GoogleNet with 1x1 and 3x3 conv layers. The network is shwon in Fig 21.

Loss function

YOLOv1's loss function is made up of two components: L_{loc}, localization loss for bounding box prediction, L_{cls}

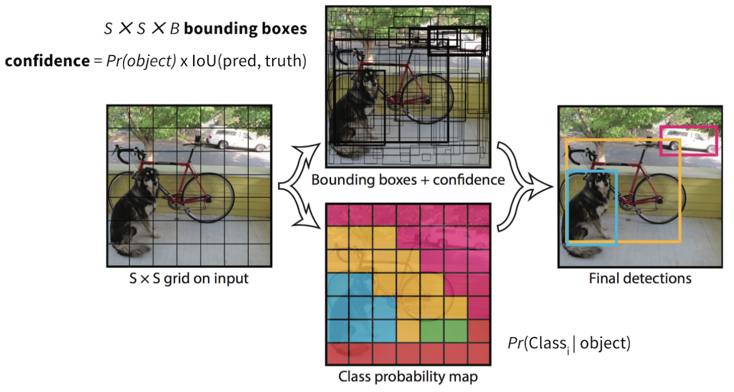


Figure 20: Visualization of the idea behind YOLOv1. Source: [36]

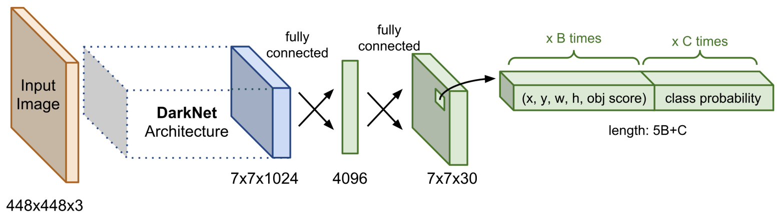


Figure 21: Visualization of YOLOv1's architecture. Source: [52]

for the class probabilities. Both of them are computed as SSE (Sum of Squared Errors). Two hyperparameters are used: λ_{coord}, which regulates the contribution to loss from bounding box coordinate predictions, and λ_{noobj}, which instead regulates the loss for boxes without objects.

$$L_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \tag{18}$$

$$L_{cls} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} (\mathbb{1}_{ij}^{obj} + \lambda_{noobj}(1 - \mathbb{1}_{ij}^{obj}))(C_{ij} - \hat{C}_{ij})^2 + \sum_{i=0}^{S^2} \sum_{c \in \mathbb{C}} \mathbb{1}_{i}^{obj} (p_i(c) - \hat{p}_i(c))^2 \tag{19}$$

$$L = L_{loc} + L_{cls} \tag{20}$$

where $\mathbb{1}_i^{obj}$ is 1 if the i^{th} cell contains an object, $\mathbb{1}_{ij}^{obj}$ is 1 if the j^{th} bounding box of the i^{th} cell is responsible for the prediction, C_{ij} is the confidence score of the i^{th} cell, $\hat{C}_{ij}$ is the predicted confidence, $\mathbb{C}$ is the set of categories, $p_i(c)$ is the conditional probability of the i^{th} cell containing an object of class c, $\hat{p}_i(c)$ is the predicted conditional probability of class c.

3.2.2 YOLOv2

YOLOv2 was introduced by Redmon et al. in 2016 [37]. Compared to YOLOv1 it presents many new features which made it improve from a mAP of 63.4 to 78.6 on the dataset VOC 2007. The changes are below listed.

Batch normalization
A batchnorm layer normalizes its inputs by applying a transformation that keeps the mean output close to 0 and the output standard deviation close to 1. This stabilizes the training process and reduces the number of epochs needed.

High Resolution Classifier
The first version of YOLO trains the CNN network at a 224x224 resolution, and then uses a 448x448 resolution for detection. YOLOv2 instead fine tunes the classifier at the 448 resolution for 10 epochs on ImageNet. This change increases the mAP by 4%.

Anchor boxes

Inspired by Faster R-CNN, YOLOv2 uses convolutional layers to predict anchor boxes. The prediction mechanisms for spatial location and probabilities are decoupled.

Dimension clusters

Instead of hand-picking the sizes of the anchor boxes, YOLOv2 uses k-mean clustering on the training data to pick better priors for the network to start from. The distance metric used is:

$$d(\text{box,centroid}) = 1 - \text{IoU}(\text{box,centroid}) \tag{21}$$

Direct Location prediction

If the detection network could place boxes in any part of the image, predicting anchor boxes would lead to unstability. To avoid this, YOLOv2 predicts location coordinates relative to the grid cell, and uses the sigmoid function to force the prediction into the range $[0, 1]$. For each box the network outputs 5 coordinates t_x, t_y, t_w, t_h, t_o. If the cell's offset from the top-left corner of the image is given by (c_x, c_y) and the bounding box prior has size (p_w, p_h), then the predicted locations are computed as:

$$b_x = \sigma(t_x) + c_x \tag{22}$$

$$b_y = \sigma(t_y) + c_y \tag{23}$$

$$b_w = p_w e^{t_w} \tag{24}$$

$$b_h = p_h e^{t_h} \tag{25}$$

$$Pr(\text{object}) * IoU(b, \text{object}) = \sigma(t_o) \tag{26}$$

The constraint placed on the predictions make the learning easier and the network more stable.

Fine-grained Features

YOLOv2 utilizes a passthrough layer to bring features from an earlier layer at resolution 26x26 to the output layer.

Multi-scale training

To make this version of YOLO flexible with respect to image sizes, the authors randomized the image size used during training: every 10 epochs the network randomly chooses an image resolution multiple of 32 from a minimum of 320 to a maximum of 608.

DarkNet-19

YOLOv2 uses a new classification network, called DarkNet-19, with 19 convolutional layers and 5 maxpooling layers.

3.2.3 YOLOv3

Introduced by Redmon et al. in 2018 [38], YOLOv3 takes many advacements made in the object detection field and incorporates them to improve from YOLOv2.

Logistic regression for confidence

While the previous versions used a Least Squares approach, YOLOv3 uses logistic regression to predict the confidence for a box.

No softmax for class prediction

Independent logistic classifiers for each class are used instead, which enables a multilabel approach. This is helpful because not all labels are guaranteed to be mutually exclusive (e.g. woman, person).

Predictions across scales

In YOLOv3, the predictions are made on 3 different scales. This is achieved by applying the same idea as FPN.

DarkNet-53

A new network as a backbone to extract features is used, called DarkNet-53, which is composed still by 3x3 and 1x1 convolutional layer but in this version there are 53 of them instead of 19. It also employs residual blocks, which are basically skip connections that link a layer's output to a deeper layer; they have been shown to fix the vanishing gradient issue.

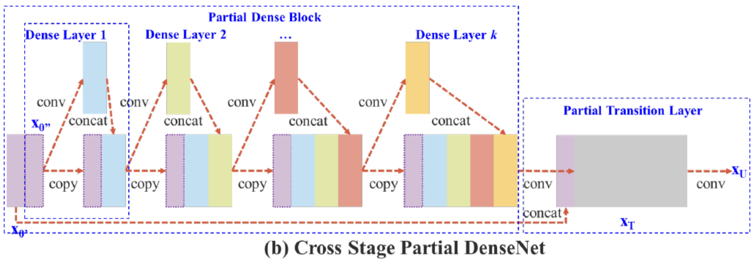


Figure 22: Structure of CSPDenseNet. Source: [50]

3.2.4 YOLOv4

Bochkovskiy et al. launched YOLOv4 in 2020 [4]. It uses Cross Stage Partial connections [50] in Darknet, giving birth to a new backbone network CSPDarkNet53. CSP is a technique which entails dividing the feature map at the base into two partitions: one goes through a dense block and a transition layer, while the other is combined with the first one at the next stage. A representation of CSP is shown in Fig 22. This addition brings several benefits: reduced vanishing gradient issues, boosted backpropagation, removal of computational bottleneck, and better learning [34].
YOLOv4 adds a SPP block: the feature maps output of CSPDarkNet53 are sent through the SPP to increase the receptive field and separate out the most important features. It also adopts the idea of Path Aggregation Network (PAN), basically an advanced version of FPN that adds a bottom-up augmentation path beside the top-down path already present in FPN.
Moreover, YOLOv4 exploits what the authors themselves call "bag of freebies" (BoF) and "bag of specials" (BoS). The former includes methods that only change the training strategy without affecting the inference cost; the latter is made up of techniques that slightly increase the inference cost, but dramatically improve the accuracy. The ones used in YOLOv4 are listed below:

- BoF for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing;
- BoS for backbone: Mish activa- tion, Cross-stage partial connections (CSP), Multi- input weighted residual connections (MiWRC);
- BoF for detection: CIoU-loss, CmBN, DropBlock regularization, Mosaic data aug- mentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyper- parameters, Random training shapes;
- BoS for detection: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS;

3.2.5 YOLOv5

Released by Glenn Gocher et al. [22] just a month after YOLOv4, YOLOv5 does not bring dramatic improvements: it still uses CSPDarkNet53, but replaces the first 3 layers with a Focus Layer, called SpaceToDepth in [41] which applies a transformation to rearrange blocks of spatial data into depth; the improvements obtained are: reduced CUDA memory, reduced layers, increased forward propagation and backpropagation. Another change is to the anchor system, which in this version has been integrated in YOLO: no datasets need to be considered, since the model automatically learns the best anchor boxes and uses them during training.
The architecture of YOLOv5 is shown in Fig 23.

3.2.6 YoloFastestV2

We want our architecture to be easily deployable on embedded devices; this entails low computational and memory requirements. The YOLO models do not satisfy these needs, and are commonly executed on normal computers or highend mobile devices. Therefore we searched for a YOLO version aimed at mobile and embedded devices. One of them is YoloFastestV1.1 [8] and the newer YoloFastestV2 [9]. The updated version is the one we adopted in our application; it brings the following improvements:

- Different loss weights for different scale output layers;
- ShuffleNetV2 as backbone;
- Anchor boxes mechanism taken from YOLOv5, and the classification loss is replaced by softmax cross entropy from sigmoid;
- Decouple the detection head, distinguish obj (foreground background classification), cls (category classification), reg (detection frame regression) 3 branches;

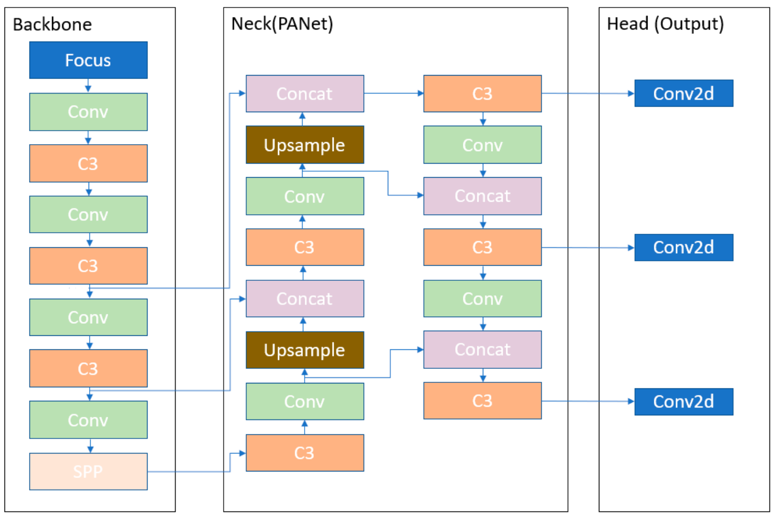


Figure 23: Architecture of YOLOv5. Source: [34]

YoloFastest benchmarks

	COCO mAP(0.5)	Run Time(1xCore)	FLOPs(G)	Params(M)
Yolo-FastestV2	24.10%	5.37 ms	0.212	0.25M
Yolo-FastestV1.1	24.40%	7.54 ms	0.252	0.35M
Yolov4-Tiny	40.2%	55.44ms	6.9	5.77M

Table 1: Benchmarks of YolofastestV2, YoloFastestV1.1, YOLOv4-Tiny tested on Mate 30 Kirin 990 CPUBased on NCNN. Source: https://github.com/dog-qiuqiu/Yolo-FastestV2

As can be seen in Table 1, YoloFastest is extremely lightweight and fast, with an inference time on 1 core that is 11 times smaller than YOLOv4-Tiny. This makes it an ideal model for performing object detection on very limited hardware.

3.3. SSD and MobileNet overview

The second detection model we used in our work is SSDLiteMobileNetV3, i.e. it applies the detection algorithm SSDLite with the CNN MobileNetV3 as a backbone. While YOLO was developed to be fast, it was still aimed at pretty high-end hardware; on the other hand in the last few years more and more attention has been given to object detection on mobile and embedded devices. MobileNet is one of the networks developed exactly for this purpose. In this section we will go over the different versions of MobileNet that have been developed throughout, as well as the difference between SSD and SSDLite.

3.3.1 MobileNet

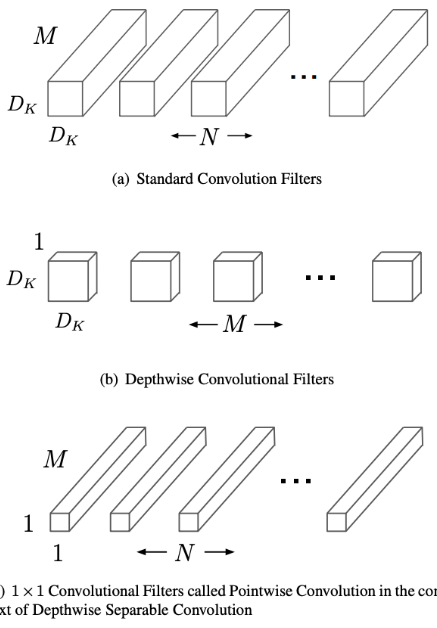
MobileNet was presented in 2017 by Howard et al. [20] and was one of the first initiatives to build CNN architectures that can easily be deployed in mobile applications. The main innovation is that it is a network built on depthwise separable convolutions.
A standard convolutional layer gets as input a $D_F \times D_F \times M$ feature map $\mathbf{F}$ and produces a $D_F \times D_F \times N$ feature map $\mathbf{G}$, where D_F is the width and height of a square input feature map, M is the number of input channels, D_G is the width and height of a square output feature map, N is the number of output channels. The kernel will have size $D_K \times D_K \times M \times N$, where D_K is the dimension of a kernel assumed to be square. The output feature map is computed as:

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \tag{27}$$

Thus the computational cost of standard convolution is:

$$D_K \times D_K \times M \times N \times D_F \times D_F \tag{28}$$

From a semantic point of view, the convolution has the effect of filtering features based on the kernel and combining them to produce a new representation.
Depthwise separable convolution splits the filtering and combining into two different layers: depthwise convolution to apply a single filter for each input channel, and pointwise convolution, simply a 1x1 convolution, to combine via linear combination the output of the depthwise convolution layer. A representation of the two types of convolution is found in Fig 24.



M
D_K
D_K
$\leftarrow N \rightarrow$
(a) Standard Convolution Filters
1
D_K
D_K
$\leftarrow M \rightarrow$
(b) Depthwise Convolutional Filters
M
1
1
$\leftarrow N \rightarrow$
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 24: Comparison between standard convolution and depthwise separable convolution. Source: [20]

Depthwise convolution with one filter per channel is computed as:

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+1-1,l+j-1,m} \tag{29}$$

and has a computational cost equal to:

$$D_K \times D_K \times M \times D_F \times D_F \tag{30}$$

Summing it up with the pointwise convolution, the total cost is:

$$D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F \tag{31}$$

Therefore the separation of the convolutional layer into two reduces the computation by a factor of $\frac{1}{N} + \frac{1}{D_K^2}$.

3.3.2 MobileNetV2

The second version of MobileNetV2, released by Howard et al. in 2018 [42], adds three major features:

- ReLU6 to speed up activations for low-precision computations and to make them more suitable for quantization. The ReLU6 is defined as:

$$\text{ReLU6}(x) = \min(\max(x, 0), 6) \tag{32}$$

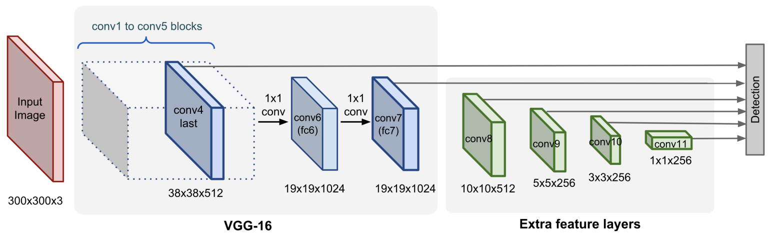


Figure 25: Architecture of SSD. Source: [52]

- Linear Bottlenecks, which are Bottleneck blocks without the last activation. The reason is that the authors have shown that the ReLU function destroys useful information when the input is < 0.
- Inverted Residual, equal to the inversion of a traditional Residual Block: first widen with a 1x1 convolution, then use a 3x3 depthwise convolution, then a 1x1 convolution to reduce the number of channels.

3.3.3 MobileNetV3

In 2019 Howard et al. proposed MobileNetV3, tuned for mobile CPUs by using Neural Architecture Search (NAS). NAS consists of automatically finding efficient neural network architectures with machine learning; the most common methods are based on Reinforcement Learning. Two NAS methods are used in MobileNetV3:

- Hardware-aware NAS, for blockwise search;
- NetAdapt for fine-tuning the single layers one by one;

Another improvement was the Hard Swish activation. It had been shown that replacing ReLU with the swish activation

$$\text{swish}(x) = x \cdot \sigma(x) \tag{33}$$

improved the accuracy of networks. However the authors of MobileNet believed the sigmoid function to be too expensive to compute om mobile hardware. For this reason they replaced the sigmoid with its piece-wise linear analog, thus obtaining the hard version of swish:

$$\text{h-swish}(x) = x\frac{\text{ReLU6}(x+3)}{6} \tag{34}$$

3.3.4 SSD

SSD is based on the backbone VGG16 trained on the ImageNet dataset, to which it adds extra convolutional layers of decreasing size. This way a pyramidal structure is obtained, where each level is used for detection; this allows the model to recognize objects of different sizes. The architecture is shown in Fig 25.

Workflow

SSD partitions the image using a grid and each grid cell is responsible for detecting objects in that region of the image. Predefined anchor boxes, each with its own size and position, are assigned to cells and the detection consists of predicting the offset of said boxes.

The anchor boxes on different levels are rescaled so that one feature map is only responsible for objects at one particular scale. The width, height, and center of the anchor boxes are normalized to be in the interval $[0, 1]$. At location (i, j) at the l^{th} feature layer of size $m \times n$ we have scales s_l that are proportional to the layer level l and to 5 different aspect ratios $r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$; In addition a special scale s'_l is used when the aspect ratio is 1, therefore we have 6 anchor boxes per cell.

$$s_l = s_min + \frac{s_{max} - s_{min}}{(L-1)}(l-1) \tag{35}$$

$$s'_l = \sqrt{s_l s_{l+1}} \text{ when } r = 1 \tag{36}$$

The width w^r_l, the height h^r_l, and the center (x^i_l, y^j_l) of the anchor box will be:

$$w^r_l = s_l\sqrt{r} \tag{37}$$

$$h^r_l = \frac{s_l}{\sqrt{r}} \tag{38}$$

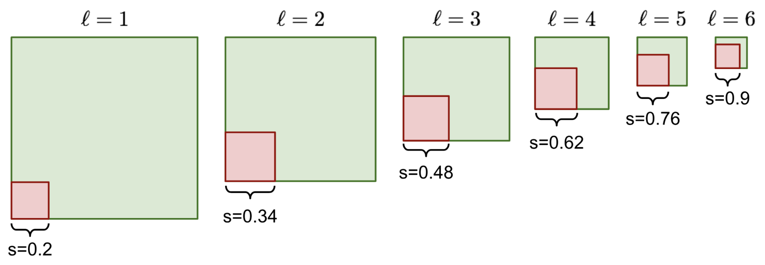


Figure 26: Anchor boxes scaling. Source: [52]

$$(x_l^i, y_l^j) = (\frac{i+0.5}{m}, \frac{j+0.5}{n}) \tag{39}$$

This means that at for every pair (i, j), the model produces 4 offsets and C class probabilities by applying a $3 \times 3 \times p$ convolution per anchor box, where p is the depth of the feature map. Thus, for a feature map $m \times n$ and k anchor boxes, SSD uses $k \times m \times n \times (C + 4)$ convolution filters.
Fig 26 shows how the anchor box of aspect ratio $r = 1$ gets scaled depending on the feature layer.

Loss function
Similarly to YOLO, the loss function is composed of two parts: localization loss and classification loss.
The localization loss is given by a smooth $\mathbb{L}_1$ loss between the predicted box and the ground truth:

$$L_{loc} = \sum_{i,j} \sum_{m \in \{x,y,w,h\}} \mathbb{1}_{ij}^{match} \mathbb{L}_1^{smooth} (d_m^i - t_m^j)^2 \tag{40}$$

$$\mathbb{L}_1^{smooth} = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{41}$$

$$t_x^j = \frac{(g_x^j - p_x^i)}{p_w^i} \tag{42}$$

$$t_y^j = \frac{(g_x^j - p_x^i)}{p_h^i} \tag{43}$$

$$t_w^j = \log(\frac{g_w^j}{p_w^i}) \tag{44}$$

$$t_h^j = \log(\frac{g_h^j}{p_h^i}) \tag{45}$$

where $\mathbb{1}_{ij}^{match}$ indicates if the i^{th} bounding box encoded by the tuple $(p_x^i, p_y^i, p_w^i, p_h^i)$ is assigned to the j^{th} ground truth box encoded by the tuple $(g_x^j, g_y^j, g_w^j, g_h^j)$, and d_m^i is the predicted correction term.
The classification loss is a softmax loss:

$$L_{cls} = -\sum_{i \in pos} \mathbb{1}_{ij}^k \log(softmax(c_i^k)) - \sum_{i \in neg} \log(softmax(c_i^0)) \tag{46}$$

where $pos = \{setofmatchedboxes\}$ and $neg = \{setofnegativeexamples\}$.
The overall loss function is obtained as:

$$L = \frac{1}{N}(L_{loc} + \alpha L_{cls}) \tag{47}$$

where N is the number of matched boxes and α is a hyperparameter that regulates the two components.

3.3.5 SSDLite

SSDLite was a variant of SSD presented together with MobileNetV2 [42]. SSDLite is specifically developed to be suitable to mobile hardware by being much more efficient. This is achieved by replacing all regular convolutions with depthwise separable convolutions which results in a great reduction of both parameter count and computational cost. The comparison between SSD and SSDLite can be seen in Table **??**.

SSD vs SSDLite

	Params	MAdds
SSD	14.8	1.25
SSDLite	2.1	0.35

Table 2: Comparison of SSD vs SSDLite. Source: [42]

Table 3 shows the comparison between the performance of SSD300, which has VGG16 as backbone, and SSDLite with the three versions of MobileNet. It is clear that while losing a small amount of accuracy, MobileNet and SSDLite bring an extreme improvement on the speed and memory side.

SSD and MobileNet benchmarks

	COCO mAP(0.5)	Params (M)	MAdds (B)	Latency (ms)
SSD300	23.2	36.1	35.2	/
MNetV1+SSDLite	22.2	5.1	1.3	228
MNetV2+SSDLite	22.1	4.3	0.80	162
MNetV3+SSDLite	22.0	4.97	0.62	137

Table 3: Benchmarks of SSD300, SSDLiteMobileNetV1, SSDLiteMobileNetv2, SSDLiteMobileNetv3 Source: [19, 42]

3.4. Distance approximation

The distance module implemented in our solution receives the array containing the detected objects in the image and, if among them there is one labeled as a person, it is able to estimate the distance he/she is standing at just by knowing the height and the focal length of the camera that took the picture.
This is done by exploiting the Thin Lens equation:

$$\frac{1}{S_2} + \frac{1}{S_1} = \frac{1}{f} \tag{48}$$

where S_1 is the distance from the object to the lens, S_2 is the distance from the lens to the image, and f is the focal length of the lens (see Fig 27). The focal length of a lens is an optical property of the lens that expresses the distance, in millimetres, between the optical centre of the lens and the camera's sensor.
The Thin Lens model is a thoretical simplified case where we consider one glass element with only one node central point; normal cameras have multiple glass elements that allow them to correct aberrations and distortions, and to zoom. Nonetheless the Thin Lens model remains valid for computational purposes.
Since the ratio of the distances on each side of the lens are the same as the ratio of the size dimension on each side of the lens, we can write the following equation:

$$\frac{S_1}{f(mm)} = \frac{H_1}{H_2} \tag{49}$$

where H_1 is the real height of the object on the sensor, H_2 is the height of the object on the sensor, S_1 is the distance of the object from the lens. It follows that we can compute the distance of the object from the camera as:

$$S_1 = \frac{f(mm) \times H_1}{H_2} \tag{50}$$

We suppose to know the height of the person we are detecting, and the focal length of the camera we are using, while the object height on the sensor needs to be computed at each detection with the following formula:

$$H_2 = \frac{SH(mm) \times OH(pixels)}{IH(pixels)} \tag{51}$$

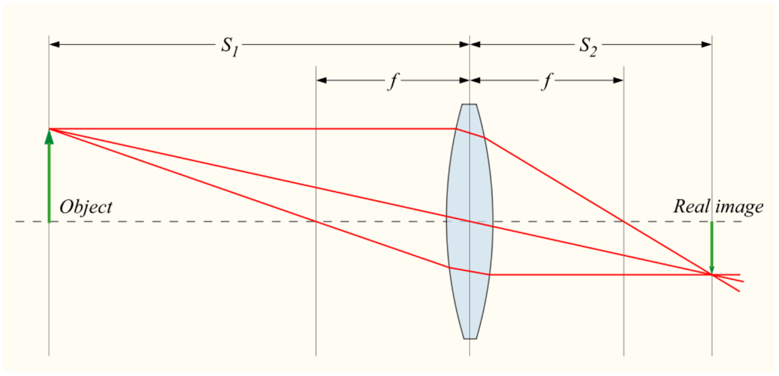


Figure 27: Thin Lens model. Source: [53]

where SH is the sensor height expressed in millimeters, a property of the camera that we suppose to know, OH is the Object Height in pixels, i.e. the height of the bounding box predicted by our detection network, and IH is the Image Height in pixels that is equal to the height resolution of the camera, if the image does not get resized. Substituting Eq 51 into Eq 50, we obtain:

$$S_1 = \frac{f(mm) \times IH(pixels) \times H_1}{SH(mm) \times OH(pixels)} \tag{52}$$

Observe that, since the focal length and the sensor height are proportional, $\frac{f(mm) \times IH(pixels)}{SH(mm)}$ is the conversion of the focal length expressed in pixels. So at last we can compute the distance of the object as:

$$S_1 = \frac{f(pixels) \times H_1}{OH(pixels)} \tag{53}$$

However, in order to reduce the computational load our application resizes the images to a 320×320 resolution before feeding them to the detection algorithm. This is an issue because now we have different pixels sizes. To understand this, consider an object 1 meter long and a 1048×1048 of it is taken, and then the image gets resized to 320×320; if in the first image the object "occupies" 400 pixels, in the second image it will take up $400 \cdot \frac{320}{1048}$ pixels. As a consequence, Eq 53 needs to be adjusted:

$$S_1 = \frac{f(pixels) \times H_1 \times 320}{OH(pixels) \times IH(pixels)} \tag{54}$$

This also means that our application is very versatile and portable; it can work with multiple cameras and deal with images at any resolution, and all it needs to know are the camera specifications and the real height of the object.

4. Dataset construction

To evaluate our solution we needed a dataset of images containing people and their distance from the camera had to be known. A dataset of this kind could not be find publicly in the literature, so we created our own. 384 pictures have been taken with the rear camera of an iPhone X in a home environment, which we believe is similar enough to an hospital environment and can be a good indicator of the performance of the algorithm. In order to get as close as possible to a real estimation, we tried to emulate 8 different environments by changing the background or the light. We have 4 different background that we called *living_room_1*, *living_room_2*, *hallway_1*, *hallway_2*; and two type of lights: *natural*, artificial. The environments can be seen in Fig 28.

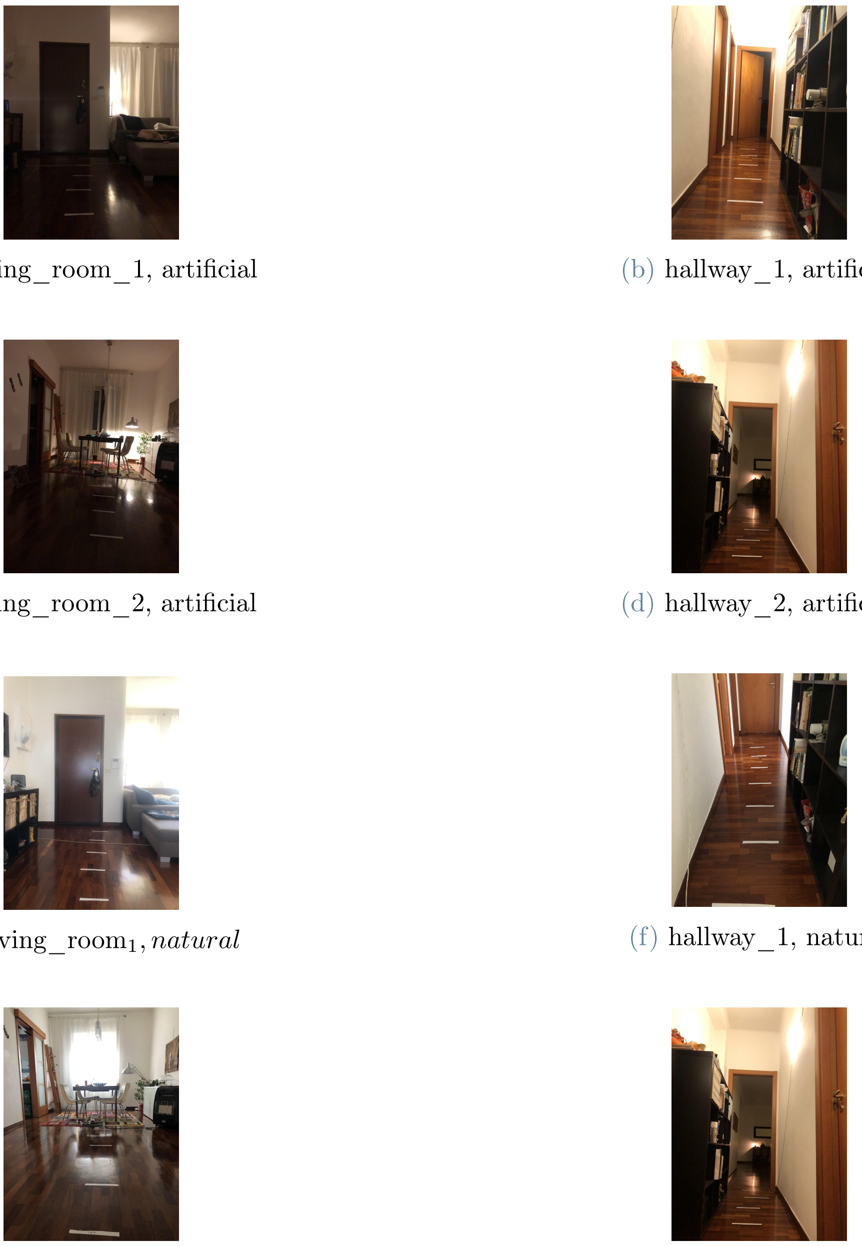

(a) living_room_1, artificial
(b) hallway_1, artificial
(c) living_room_2, artificial
(d) hallway_2, artificial
(e) living_room$_1$, *natural*
(f) hallway_1, natural
(g) living_room_2, natural
(h) hallway_2, natural

Figure 28: The 8 different environments that were used for the creation of the dataset.

The photos either contain no people or only one person that is standing straight facing different directions and at one of the distances from the following set {1.5m,2.0m,2.5m,3.0m,3.5m}; 5 different distances, and the empty case, sum up to 6 different scenarios per environment, and for each of these 8 pictures were taken.

5. Experimental results

The following tables show respectively the performance achieved by our algorithm when using YoloFastestV2 and SSDLiteMobileNetV3 as the detection module.

YoloFastestV2 performance

Distance	RMSE	Max Error	Min Error	FP Rate	FN Rate	Environment
N/A				0.000	0.000	living_room_1 artificial
1.5	**0.181m**	0.274m	0.028m	0.000	0.000	
2	**0.102m**	0.166m	0.016m	0.000	0.000	
2.5	**0.094m**	0.218m	0.008m	0.000	0.125	
3	**0.222m**	0.546m	0.018m	0.000	0.125	
3.5	**0.127m**	0.210m	0.004m	0.000	0.000	
avg	**0.145m**	0.283m	0.015m	0.000	0.042	
N/A				0.000	0.000	hallway_1_artificial
1.5	**0.200m**	0.293m	0.028m	0.125	0.000	
2	**0.097m**	0.166m	0.008m	0.000	0.000	
2.5	**0.080m**	0.185m	0.005m	0.000	0.000	
3	**0.468m**	1.118m	0.018m	0.000	0.125	
3.5	**0.541m**	0.958m	0.267m	0.000	0.125	
avg	**0.277m**	0.544m	0.065m	0.021	0.042	
N/A				0.000	0.000	living_room_2 artificial
1.5	**0.285m**	0.467m	0.058m	0.000	0.000	
2	**0.211m**	0.479m	0.077m	0.000	0.000	
2.5	**0.522m**	0.947m	0.058m	0.125	0.000	
3	**0.747m**	1.718m	0.135m	0.000	0.375	
3.5	**1.112m**	2.426m	0.022m	0.000	0.125	
avg	**0.575m**	1.207m	0.070m	0.021	0.083	
N/A				0.000	0.000	hallway_2 artificial
1.5	**0.176m**	0.261m	0.109m	0.000	0.000	
2	**0.124m**	0.199m	0.033m	0.000	0.000	
2.5	**0.221m**	0.445m	0.005m	0.125	0.000	
3	**0.270m**	0.447m	0.019m	0.000	0.000	
3.5	**0.202m**	0.451m	0.004m	0.125	0.000	
avg	**0.199m**	0.361m	0.034m	0.042	0.000	
N/A				0.000	0.000	living_room_1 natural
1.5	**0.092m**	0.170m	0.038m	0.000	0.000	
2	**0.122m**	0.250m	0.016m	0.000	0.000	
2.5	**0.118m**	0.271m	0.005m	0.000	0.000	
3	**0.092m**	0.156m	0.000m	0.125	0.000	
3.5	**0.204m**	0.344m	0.004m	0.000	0.000	
avg	**0.126m**	0.238m	0.013m	0.021	0.000	
N/A				0.000	0.000	hallway_1 natural
1.5	**0.171m**	0.267m	0.043m	0.125	0.000	
2	**0.171m**	0.336m	0.033m	0.125	0.000	
2.5	**0.285m**	0.450m	0.021m	0.000	0.000	
3	**0.132m**	0.207m	0.019m	0.000	0.000	
3.5	**0.266m**	0.516m	0.073m	0.000	1.000	
avg	**0.205m**	0.355m	0.038m	0.042	0.167	
N/A				0.125	0.000	living_room_2 natural
1.5	**0.076m**	0.170m	0.009m	0.000	0.000	
2	**0.172m**	0.246m	0.094m	0.000	0.000	
2.5	**0.186m**	0.321m	0.031m	0.000	0.000	
3	**0.295m**	0.558m	0.000m	0.000	0.000	
3.5	**0.290m**	0.763m	0.029m	0.000	0.000	
avg	**0.204m**	0.412m	0.033m	0.021	0.000	
N/A				0.000	0.000	hallway_2 natural
1.5	**0.097m**	0.170m	0.048m	0.000	0.000	
2	**0.096m**	0.180m	0.000m	0.000	0.000	
2.5	**0.257m**	0.441m	0.070m	0.000	0.000	
3	**0.387m**	0.582m	0.055m	0.000	0.000	
3.5	**0.504m**	0.887m	0.171m	0.000	0.000	
avg	**0.268m**	0.452m	0.069m	0.000	0.000	

Table 4: Results obtained by running the application with YoloFastestV2 as the detection module on different environments.

SSDLiteMobileNetV3 performance

Distance	RMSE	Max Error	Min Error	FP Rate	FN Rate	Environment
N/A				0.000	0.000	living_room_1 artificial
1.5	**0.105m**	0.287	0.028	0.000	0.000	
2	**0.309m**	0.467	0.207	0.000	0.000	
2.5	**0.577m**	0.645	0.525	0.000	0.000	
3	**0.709m**	1.048	0.641	0.000	0.250	
3.5	**1.185m**	1.458	1.008	0.000	0.125	
avg	**0.577m**	0.781	0.4818	0.000	0.063	
N/A				0.000	0.000	hallway_1_artificial
1.5	**0.066m**	0.142	0.028	0.000	0.125	
2	**0.345m**	0.369	0.301	0.000	0.000	
2.5	**0.491m**	0.602	0.35	0.000	0.000	
3	**0.544m**	0.821	0.482	0.000	0.250	
3.5	**1.150m**	1.45	0.844	0.000	0.125	
avg	**0.519m**	0.6768	0.401	0.000	0.083	
N/A				0.000	0.000	living_room_2 artificial
1.5	**0.102m**	0.248	0.028	0.000	0.125	
2	**0.314m**	0.427	0.102	0.000	0.000	
2.5	**0.233m**	0.359	0.175	0.000	0.250	
3	**0.496m**	0.75	0.344	0.000	0.125	
3.5	**0.897m**	1.281	0.607	0.000	0.000	
avg	**0.408m**	0.613	0.2512	0.000	0.083	
N/A				0.000	0.000	hallway_2 artificial
1.5	**0.042m**	0.073	0.028	0.000	0.000	
2	**0.343m**	0.407	0.18	0.000	0.125	
2.5	**0.460m**	0.579	0.414	0.000	0.000	
3	**0.795m**	1.109	0.686	0.000	0.000	
3.5	**1.222m**	1.414	0.887	0.000	0.000	
avg	**0.572m**	0.7164	0.439	0.000	0.021	
N/A				0.000	0.000	living_room_1 natural
1.5	**0.033m**	0.063	0.028	0.000	0.125	
2	**0.316m**	0.391	0.145	0.000	0.000	
2.5	**0.486m**	0.602	0.484	0.000	0.250	
3	**0.722m**	1.064	0.652	0.000	0.125	
3.5	**0.908m**	1.405	0.77	0.000	0.250	
avg	**0.493m**	0.705	0.4158	0.000	0.125	
N/A				0.000	0.000	hallway_1 natural
1.5	**0.064m**	0.12	0.028	0.000	0.125	
2	**0.347m**	0.467	0.258	0.000	0.000	
2.5	**0.557m**	0.7	0.533	0.000	0.125	
3	**0.580m**	0.811	0.533	0.000	0.250	
3.5	**1.408m**	1.516	1.33	0.000	0.000	
avg	**0.591m**	0.7228	0.5364	0.000	0.083	
N/A				0.000	0.000	living_room_2 natural
1.5	**0.028m**	0.028	0.028	0.000	0.000	
2	**0.352m**	0.407	0.233	0.000	0.000	
2.5	**0.548m**	0.673	0.281	0.000	0.125	
3	**0.621m**	1.04	0.344	0.000	0.125	
3.5	**1.030m**	1.281	0.859	0.000	0.000	
avg	**0.516m**	0.6858	0.349	0.000	0.042	
N/A				0.000	0.000	hallway_2 natural
1.5	**0.047m**	0.109	0.028	0.000	0.000	
2	**0.282m**	0.369	0.048	0.000	0.000	
2.5	**0.540m**	0.638	0.492	0.000	0.125	
3	**0.864m**	1.124	0.878	0.000	0.250	
3.5	**1.248m**	1.475	1.033	0.000	0.125	
avg	**0.596m**	0.743	0.4958	0.000	0.083	

Table 5: Results obtained by running the application with SSDMobileNetV3 as the detection module on different environments.

Confusion Matrices on all envs

Table 6: YoloFastestV2

	Predicted one	Predicted zero	Predicted more
Actual one	304	9	7
Actual zero	1	63	0

Table 7: SSDLiteMobileNetV3

	Predicted one	Predicted zero	Predicted more
Actual one	292	28	0
Actual zero	0	64	0

- YoloFastestV2 - Precision = 97.44%, Recall = 95.00%, F-score = 96.20%
- SSDLiteMobileNetV3 - Precision = 100.00%, Recall = 91.25%, F-score = 95.42%

Confusion Matrices with natural light

Table 8: YoloFastestV2

	Predicted one	Predicted zero	Predicted more
Actual one	156	1	3
Actual zero	1	31	0

Table 9: SSDLiteMobileNetV3

	Predicted one	Predicted zero	Predicted more
Actual one	144	16	0
Actual zero	0	32	0

- YoloFastestV2 - Precision = 97.50%, Recall = 97.50%, F-score = 97.50%
- SSDLiteMobileNetV3 - Precision = 100.00%, Recall = 90.00%, F-score = 94.74%

Confusion Matrices with artificial light

Table 10: YoloFastestV2

	Predicted one	Predicted zero	Predicted more
Actual one	148	8	4
Actual zero	0	32	0

Table 11: SSDLiteMobileNetV3

	Predicted one	Predicted zero	Predicted more
Actual one	148	12	0
Actual zero	0	32	0

- YoloFastestV2 - Precision = 97.37%, Recall = 92.50%, F-score = 94.87%
- SSDLiteMobileNetV3 - Precision = 100.00%, Recall = 92.50%, F-score = 96.10%

YoloFastestV2 causes some false-positives but it performs very well when a person is actually present in the picture; on the other hand, SSDLiteMobileNetV3 never reports a person when nothing is present, but sometimes it misclassifies objects as people.

Here we report only the distance estimation results.

Distance estimation with YoloFastestV2 performance, RMSE(m)

Distance	All	Natural	Artificial
1.5	0.160	0.109	0.211
2	0.137	0.140	0.134
2.5	0.220	0.212	0.229
3	0.327	0.227	0.427
3.5	0.406	0.316	0.496
avg	0.250	0.201	0.299

Distance	All	living_room_1	living_room_2	hallway_1	hallway_2
1.5	0.160	0.137	0.181	0.186	0.137
2	0.137	0.112	0.192	0.134	0.110
2.5	0.220	0.106	0.354	0.183	0.239
3	0.327	0.157	0.521	0.300	0.329
3.5	0.406	0.166	0.701	0.404	0.353
avg	0.250	0.135	0.390	0.241	0.233

Table 12: Benchmarks of our algorithm when using YoloFastestV2 for detection, with RMSE at different distances, light conditions and environments

Distance estimation with YoloFastestV2 performance, RMSE(m)

Distance	All	Natural	Artificial
1.5	0.061	0.043	0.079
2	0.326	0.324	0.328
2.5	0.487	0.533	0.440
3	0.666	0.697	0.636
3.5	1.131	1.149	1.114
avg	0.534	0.549	0.519

Distance	All	living_room_1	living_room_2	hallway_1	hallway_2
1.5	0.061	0.069	0.065	0.065	0.045
2	0.326	0.313	0.333	0.346	0.313
2.5	0.487	0.532	0.391	0.524	0.500
3	0.666	0.716	0.559	0.562	0.830
3.5	1.131	1.047	0.964	1.279	1.235
avg	0.534	0.535	0.462	0.555	0.584

Table 13: Benchmarks of our algorithm when using YoloFastestV2 for detection, with RMSE at different distances, light conditions and environments

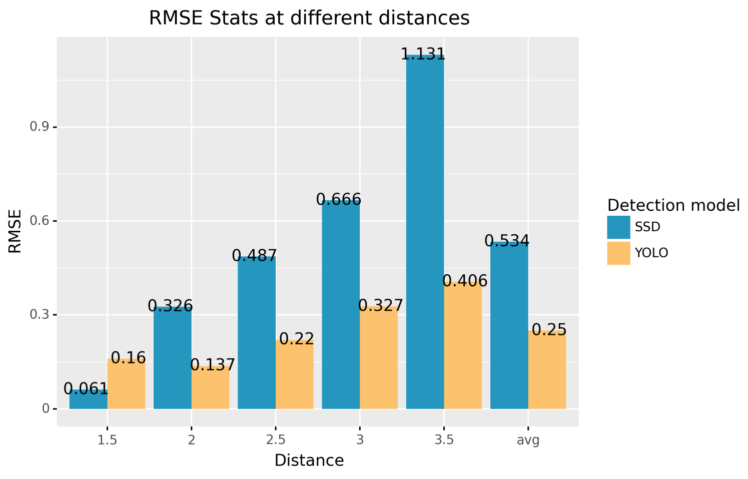


Figure 29: RMSE of the distance prediction at different distances, and the average.

Figure 29 shows the RMSE of the distance estimate with respect to the ground truth achieved at different distances with both versions of the algorithm. We can see that with both detection models the distance estimate gets worse when the actual distance is larger. Moreover YoloFastestV2 performs more than twice as better as SSDLiteMobileNetV3, except when the person is standing at 1.5 meters from the camera. We suppose this is because the way SSDMLiteobileNetV3 predicts boxes works better when the subject occupies basically the whole view, while YoloFastestV2 faces some issue in that situation.

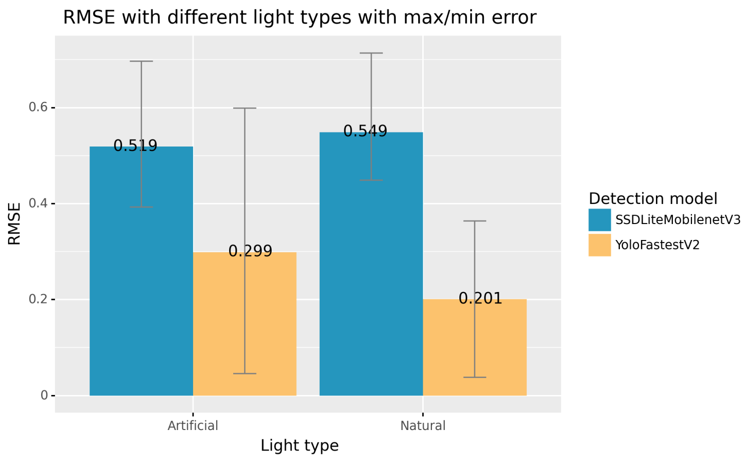


Figure 30: RMSE of the distance prediction performed in different light conditions.

Figure 30 shows the performance of the algorithm when dealing with different light conditions. SSDLiteMobileNetV3's estimate stays close to constant when changing light conditions, while YoloFastestV2 falls off with natural light.

Fig 31 shows how the performance of the estimate is affected by the environment, i.e the background of the image behind the human. Surprisingly, the two detection models have almost an opposite behavior: SSDLiteMobileNetV3 has pretty consistent performance and is best in *living_room_2*, while YoloFastestV2 has fairly high variance performs much worse in this case, while performing the best in *living_room_1*.
We suppose that *living_room_2* is the worst environment because it has the most complex background, with windows, tables, and chairs behind the subject; the detection model has a hard time detecting the whole body correctly, and the bounding box either cuts out the part of the body near the complex background, or includes in the box something that should not. An example is shown in Fig 32.
living_room_1 is probably the "simplest" environment, with almost no objects and just a wall and a door behind the subject; so the algorithm has an easy time correctly predicting the bounding box for a person. *hallway_1* and *hallway_2* are very similar to each other, hence the very close results. They are not as complex as *living_room_2*, but contain more irregularities than *living_room_1*, e.g: bookshelf on the side, doors on the side, narrow space.

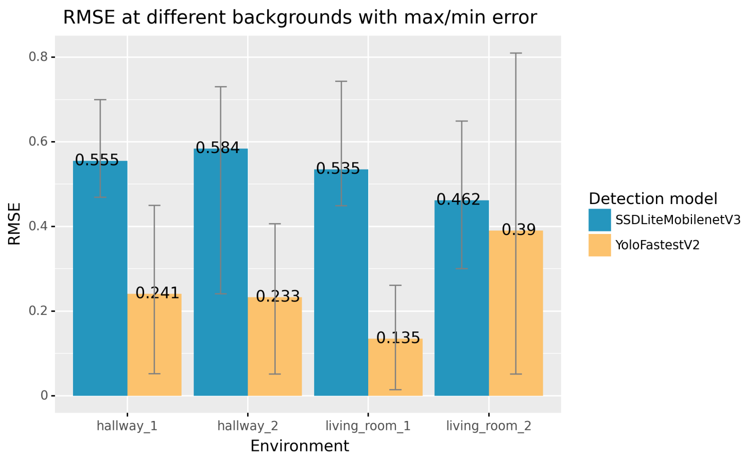


Figure 31: RMSE of the distance prediction performed in the different environments

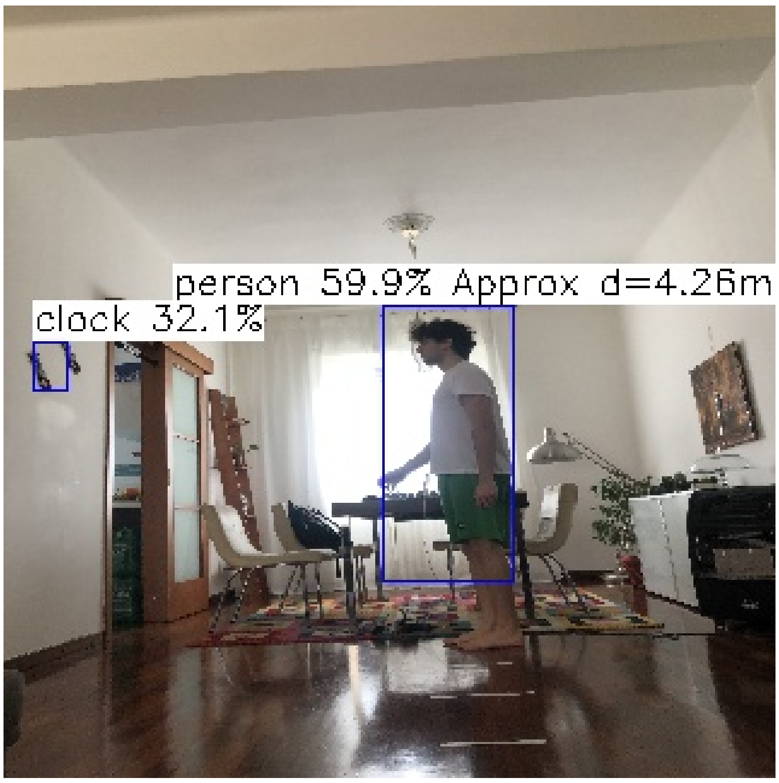


Figure 32: Example of a bad bounding box estimation in *living_room_2*.

From these results, we conclude that to maximize the performance of the algorithm choosing correct light conditions and environments is crucial. The element that most affected the algorithm was light sources behind the subject, which projected its shadow on the floor and made the detection model overestimate the size of the bounding box. We believe the ideal case would be a room with not many items or furnishing, ideally plain walls, and supplemented with lamps when the surroundings are not correctly illuminated.

6. Deployment

The proposed algorithm was tested for live object detection and distance estimation to analyze how well it would perform in a real-life scenario.
The model was implemented on a RaspberryPi 3 B+, that is equipped with a Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz, and a 1GB LPDDR2 SDRAM; the microprocessor ran on the Raspberry Pi OS 64 bit Bullseye. In addition, to take the pictures, the device was supplemented with a Raspberry Pi Camera Module V2. The setup is shown in Fig 33.

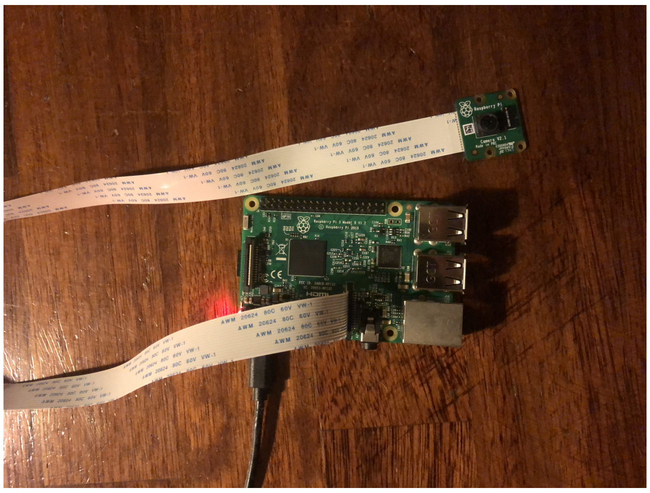


Figure 33: RMSE of the distance prediction performed in the different environments

Both SSDLiteMobileNetV3 and YoloFastestV2 were implemented using NCNN [46], a high-performance neural network inference computing framework optimized for mobile platforms. This allowed us to reach the best speed. To operate the camera module, instead, we used LCCV [24], a small wrapper library that provides access to the Raspberry Pi camera in OpenCV.
Table 14 shows how the two models compare on the Raspberry. YoloFastestV2 is clearly superior in both speed and memor space, while also achieving better results in the distance estimation.

Model	FPS (Frames Per Second)	Size (MB)
YoloFastestV2	7.5	6.2
SSDLiteMobileNetV3	5.0	15.0

Table 14: Frame per second and memory space occupied by YoloFastestV2 and SSDLiteMobileNetV3 when ran on a RaspberryPi 3 B+.

7. Conclusions

In this thesis work, we have introduced and defined both mathematically and semantically the object detection task and the main approaches that have been developed in the last decades. We gave a brief overview on the building blocks of Deep Neural Networks, specifically those that are used in Detection Networks, and we have analyzed the architectures of two of the major One-Stage Detectors nowadays: YOLO and SSD.
Then we proposed our application and its architecture, showing how its components work together, and the theory behind the distance estimation based only on the image and the camera hardware specifications. The database and the creation process by ourselves was described, and the results obtained by evaluating the application on it were shown and commented.
Lastly we described how the application was tested on a RaspberryPi 3 B+ and its perfomance.

We believe our solution works fairly well, achieving good results while being extremely lightweight and portable; it could be used in real life scenarios.

The application does however have some minor flaws: the real height of the subject must be known and that may not always be possible; to ensure optimal performance, it is needed to have a proper setup in order to avoid errors, such as those related to shadows and camera angles.

In the future, in our opinion the work could certainly be extended and improved. Some possible ideas are:

- The speed of the application could be increased by overclocking the device, provided that a quality cooling system is provided.
- Experimenting with stereo cameras to estimate the distance.
- Expanding and annotating the dataset in a more formal way could be very useful for future experiments.
- Utilizing the application to annotate a radar dataset: the output of our solution will be used to automatically label the data produced by the radar.

References

[1] Computer vision in agtech. [Online; accessed April 16, 2022].

[2] What is computer vision? [Online; accessed April 16, 2022].

[3] Object detection: Current and future directions. *Frontiers in Robotics and AI*, 2, 2015.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[5] Gaudenz Boesch. Object detection in 2022: The definitive guide, 2022. [Online; accessed April 16, 2022].

[6] Chinmoy Borah. Evolution of object detection. [Online; accessed May 3, 2022].

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

[8] dog qiuqiu. dog-qiuqiu/yolo-fastest: yolo-fastest-v1.1.0, July 2021.

[9] dog qiuqiu. dog-qiuqiu/yolo-fastestv2: V0.2, August 2021.

[10] Frank Emmert-Streib, Zhen Yang, Han Feng, Shailesh Tripathi, and Matthias Dehmer. An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence*, 3, 2020.

[11] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks, 2013.

[12] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[13] Pedro F. Felzenszwalb, Ross B. Girshick, and David McAllester. Cascade object detection with deformable part models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2241–2248, 2010.

[14] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[15] Lex Fridman. Mit deep learning basics: Introduction and overview with tensorflow. [Online; accessed May 5, 2022].

[16] Ross Girshick. Fast r-cnn, 2015.

[17] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014.

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[20] 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, 2017.

[21] Hwang, Ji-Hee, Hyun-Ji Kim, Heejin Park, Byoung-Seok Lee, Hwa-Young Son, Yong-Bum Kim, Sang-Yeop Jun, Jong-Hyun Park, Jaeku Lee, and Jae-Woo Cho. Implementation and practice of deep learning-based instance segmentation algorithm for quantification of hepatic fibrosis at whole slide level in sprague-dawley rats. *Toxicologic Pathology 50*, 2022.

[22] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, February 2022.

[23] Kushsairy Abdul Kadir, Mohd Khairi Kamaruddin, Haidawati Md Nasir, Sairul I. Safie, and Zulkifli Abdul Kadir Bakti. A comparative study between lbp and haar-like features for face detection using opencv. *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*, pages 335–339, 2014.

[24] kbarni. libcamera bindings for opencv. `https://github.com/kbarni/LCCV`, 2021.

[25] Kiprono Elijah Koech. Confusion matrix for object detection. [Online; accessed May 3, 2022].

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[27] Rasa Kundrotaitė. The evolution of object detection, 2021. [Online; accessed April 19, 2022].

[28] Harshall Lamba. Understanding semantic segmentation with unet. [Online; accessed April 17, 2022].

[29] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints, 2018.

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[31] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[33] Mahyar Najibi, Mohammad Rastegari, and Larry S. Davis. G-cnn: an iterative grid based object detector, 2015.

[34] Upesh Nepal and Hossein Eslamiat. Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs. *Sensors*, 22(2), 2022.

[35] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollar. Learning to segment object candidates, 2015.

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[37] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[38] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[39] J. Ren and Y. Wang. Overview of object detection algorithms using convolutional neural networks, 2022.

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[41] Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture, 2020.

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018.

[43] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. pages 92–101, 01 2010.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[45] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2553–2561, Red Hook, NY, USA, 2013. Curran Associates Inc.

[46] tencent. ncnn. `https://github.com/Tencent/ncnn`, 2021.

[47] Sik-Ho Tsang. Review: Fpn — feature pyramid network (object detection). [Online; accessed May 4, 2022].

[48] Koen E. A. van de Sande, Jasper R. R. Uijlings, Theo Gevers, and Arnold W. M. Smeulders. Segmentation as selective search for object recognition. In *2011 International Conference on Computer Vision*, pages 1879–1886, 2011.

[49] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[50] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[51] Donglai Wei, Zudi Lin, Daniel Franco-Barranco, Nils Wendt, Xingyu Liu, Wenjie Yin, Xin Huang, Aarush Gupta, Won-Dong Jang, Xueying Wang, Ignacio Arganda-Carreras, Jeff W. Lichtman, and Hanspeter Pfister. Mitoem dataset: Large-scale 3d mitochondria instance segmentation from em images. In Anne L. Martel, Purang Abolmaesumi, Danail Stoyanov, Diana Mateus, Maria A. Zuluaga, S. Kevin Zhou, Daniel Racoceanu, and Leo Joskowicz, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*, pages 66–76, Cham, 2020. Springer International Publishing.

[52] Lilian Weng. Object detection part 4: Fast detection models. *lilianweng.github.io*, 2018.

[53] Wikipedia contributors. Lens — Wikipedia, the free encyclopedia, 2022. [Online; accessed 6-May-2022].

[54] Xiongwei Wu, Doyen Sahoo, and Steven C. H. Hoi. Recent advances in deep learning for object detection, 2019.

[55] Donggeun Yoo, Sunggyun Park, Joon-Young Lee, Anthony S. Paek, and In So Kweon. Attentionnet: Aggregating weak directions for accurate object detection, 2015.

[56] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2019.

Abstract in lingua italiana

L'Object Detection è uno dei principali task nel contesto della Computer Vision, e può essere applicato in qualunque campo sia per migliorare meccanismi esistenti, sia per crearne di nuovi. L'obiettivo di questo lavoro di tesi è ottenere un algoritmo in grado di fare detection di persone e stimarne la distanza, ma allo stesso tempo essere leggero, portatile, e economico.
In questo scritto, prima viene data una definizione formale di Object Detection, e viene descritta l'incredibile evoluzione che ha avuto nelle ultime decadi, concentrandosi sugli approci basati su reti neurali. In seguito, viene descritta la nostra soluzione: un'applicazione basata su due modelli avanzati di Object Detection YoloFastestV2 e SSDLiteMobileNetV3, specializzati per dispositivi piccoli e con hardware limitato. Inoltre presentiamo un dataset creato da noi stessi, che è stato usato per valutare il nostro algoritmo sia dal punto di vista della detection che da quello della stima della distanza.
I risultati ottenuti suggeriscono che l'ambiente in cui viene usata l'applicazione ha un impatto significativo sulle performance, e per questo è un fattore da considerare quando si impiega questo tipo di algoritmo.

Parole chiave: object detection, stima della distanza, intelligenza artificiale, tinyML

Acknowledgements

Desidero ringraziare il Prof. Manuel Roveri per avermi dato la possibilità di realizzare questo lavoro. La sua guida e i suoi consigli sono state fondamentali per il conseguimento di questa ricerca.

Inoltre vorrei ringraziare il Dott. Massimo Pavan, per i suggerimenti durante la fase di ricerca e di scrittura e per la sua disponibilità.

E per ultimo, ringrazio Anna.