POLITECNICO DI TORINO

Faculty of Biomedical Engineering

Master Degree Thesis

Deep Learning based Detection and Tracking of the IVC in Ultrasound Scans

Supervisor
Prof. Luca Mesin

Co-supervisor:
Piero Policastro

Candidate:
Roberta Stellino

DICEMBRE 2023

Summary

This work of thesis originated within a start-up company, VIPER s.r.l., which was established as a research project of the Polytechnic of Turin, which developed a semi-automatic software to delineate and trace the displacements of the edges of the Inferior Vena Cava (IVC) not in a single direction, but in both transverse and longitudinal sections in ultrasound videos, meaning in a totally non-invasive way. This software allows the computation of different measurements, such as diameters, areas, Caval Index (CI) and so on, based on the assessment of the pulsatility of the IVC itself. These measurements can lead to the evaluation of the volemic state of a patient and the pressure present in the right atrium (Right Atrial Pressure, RAP), which are two different indicators that can be valuable in many fields of medicine, ranging from internal medicine, in order to distinct between hypovolaemic and euvolemic patients, to cardiology, for the evaluation and management of heart failure, and many more. One limitation of this software is the impossibility of obtaining these measurements in real-time: in fact, at first the ultrasound video is obtained and only in post-processing the video can be elaborated, and all the measurements of interest therefore obtained. Furthermore, as previously said, it is a semi-automatic software, meaning it needs the intervention of an operator to start working. This thesis arose out of the need to make this software work in real-time and to make it completely operator-independent. To make it possible, it was studied a deep learning methodology that allows the automatic detection of the IVC in real-time. In particular, it was chosen to use a YOLOv8 model, which is the current state-of-the-art of the real-time object detection models: the aim of this tool is to identify and classify Inferior Vena Cava within ultrasound videos, both in transverse and longitudinal sections with high accuracy. Furthermore, in order to improve the ability of the object detection model to follow the IVC movements throughout the ultrasound video, it was chosen to couple it with an object tracking model, in particular it was opted for DeepSORT, a very popular object tracking algorithm, also based on deep learning. Therefore, the discussion will begin

with an initial descriptive part that will cover anatomy and physiology of the IVC, ultrasounds basics, and an overview about object detection and the most famous algorithms used nowadays. In the second part, they will be discussed the methods that were used in this work of thesis, in particular the ones concerning YOLOv8 and DeepSORT will be explored in detail. Finally, they will be exposed the results achieved in this work of thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Inferior Vena Cava (IVC) is one of the main vases of human body and, thanks to the evaluation of its pulsatility, it can be used to obtain information about the patient's volemic state (i.e. the volume of blood present in the circulatory system) and the pressure present in the right atrium of the heart (Right Atrial Pressure - RAP). The knowledge of these two pieces of information is crucial in different areas of medicine, such as in internal medicine (for the distinction between hypovolaemic and euvolemic patients), general medicine (for the correct management of dehydration in haemodialysis therapies), cardiology (for the assessment and management of heart failure) or in pulmonology (in patients with pulmonary hypertension) [1]. Currently, the volemic status is assessed by: clinical assessment (unsatisfactory, especially in the case of hypovolaemia), invasive methods (typical in intensive care), non-invasive methods (which include ultrasound of the IVC, lung ultrasound, passive leg raising and bioimpedance testing - but mainly for research purposes). With regard to RAP, the golden standard is atrial catheterisation (a very invasive methodology), but it can also be assessed by ultrasound scanning of the IVC. The main parameter derived from ultrasound scanning of the IVC is a pulsatility index, i.e. the Caval Index (CI), which depends on the maximum diameter and the minimum diameter that the IVC assumes during a respiratory cycle: these are determined by the physician when observing the ultrasound video. It is therefore clear that the technique has several problems: the vein moves with respect to the fixed radius that is identified; it has a complicated geometry so that using only one radius can be limiting; furthermore, the temporal dynamics of the vein is complicated and cannot be described by a single measurement [1]. Finally, this methodology is highly operator-dependent and gives inaccurate and unreliable results.

This is the background for the start-up company called Vein Image Processing for Edge Rendering (VIPER) s.r.l. [2], which has developed a semi-automatic software to delineate and trace the displacements of the IVC edges, not in a single direction, but in both transverse and longitudinal sections [3]. By identifying the edges of the vein, more accurate diameter measurements can be obtained. Furthermore, working in two directions can make the diameter estimate more stable and therefore more reliable. The semi-automatic approach also makes it possible to decrease intra- and inter-operator dependency. A limitation of this software is the impossibility of obtaining the required measurements in real-time: in fact, at first the ultrasound video is obtained and only in post-processing the video can be elaborated and all the measurements of interest, such as diameters, areas, Caval Index and so on, can be obtained.

The aim of this thesis is therefore to develop, following a study of the state of the art of Object Detection methods, a Deep Learning model for the automatic detection of the inferior vena cava in real-time. In this way, the method will also be usable by less experienced operators; it will make the estimation of the volemic state and RAP more reliable, as the method could be standardised; and it will make it possible to reduce hospitalisation times (in fact, following incorrect fluid therapy caused, for example, by a gross assessment of the volemic state, a longer period of hospitalisation follows) [1].

To satisfy this goal, it was chosen to rely on the YOLO family, which includes a series of models for object detection that in the recent years led the state-of-the-art algorithms in this field. In particular, it was used the last model, which is YOLOv8, released in 2023 and yet is the current best performer model in the object detection field, in terms of speed and accuracy. The aim of this tool is to identify and classify Inferior Vena Cava within ultrasound scans, both in transverse and longitudinal sections with high accuracy. Furthermore, in order to improve the ability of the object detection model to follow the IVC movements throughout the ultrasound video, it was chosen to couple it with an object tracking model, in particular it was opted for DeepSORT, a very popular object tracking algorithm, also based on deep learning. The latter is characterized by a high level of tracking accuracy and by the ability to handle challenging scenarios, like occlusions by other anatomical structures, or suboptimal image quality that can characterize the ultrasound videos, and so on. Furthermore, it is a very fast algorithm, such that it can easily work in real-time, without affecting the speed of detection of YOLOv8. Therefore, the discussion will begin with an initial descriptive part that will cover anatomy and physiology of the IVC, ultrasounds basics,

and an overview about object detection and the most famous algorithms used nowadays. In the second part, they will be discussed the methods that were used in this work of thesis, in particular the ones concerning YOLOv8 and DeepSORT will be explored in detail. Finally, they will be exposed the results achieved in this work of thesis.

1.1 Anatomy and physiology of the IVC

The inferior vena cava is the largest venous trunk in the entire human body and is responsible for transporting oxygen-poor blood from the lower part of the body, i.e. from the lower limbs and all organs below the diaphragm, to the heart. The inferior vena cava has an average length of 22 centimetres (18 of which run in the abdomen) and a diameter of about 30 millimetres [4]. Like arteries, veins have a certain pulsatility too, mainly due to the transmural pressure P_{tm}, defined as the difference between the pressure inside and the pressure outside the vessel:

$$P_{tm} = P_{int} - P_{ext} \tag{1.1}$$

As the transmural pressure changes, there are changes in vessel volume, according to the non-linear trend shown in figure 1.1:

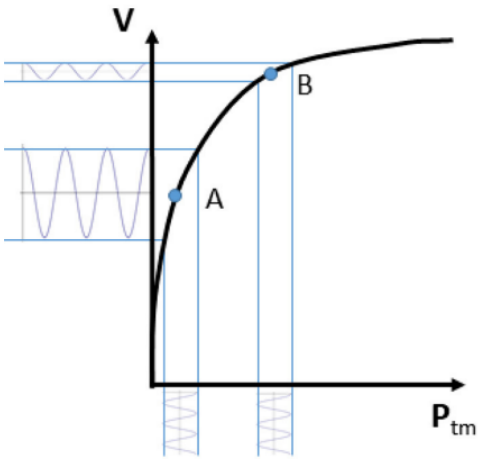


Figure 1.1: Representation of how the vessel volume changes with respect to the transmural pression [5].

The reported trend suggests that the same change in transmural pressure corresponds to different changes in volume, depending on the initial condition at which the vessel is located: at low initial volumes (point A in figure 1.1),

a large oscillation in volume - and thus in vessel size - is observed, while at large vessel diameters (point B in figure 1.1), the oscillation visibly decreases. This oscillation is the pulsatility of the vessel [5]. The variation in vessel size is due not only to vessel compliance (i.e. the ability of the vessel to expand as pressure changes), but also to the compliance of the tissues surrounding it, since, as it expands, the vessel will occupy a space that normally belongs to the other tissues and therefore depends on the compliance of these tissues.

The inferior vena cava also shows a certain pulsatility during a respiratory cycle and it also depends on the type of inspiration that is performed, as is shown in figure 1.2. If inspiration is purely thoracic, the expansion of the thorax lowers the internal pressure and blood is recalled to the thorax, thus causing a decrease in pressure and volume in the IVC. If the inspiration is diaphragmatic, nothing changes at the thoracic level, i.e. there is still a lowering of internal pressure and blood recall. What does change is that the lowering of the diaphragm increases abdominal pressure: there is an increase in external pressure that further squeezes the vein, thus causing greater pulsatility of the IVC compared to thoracic inspiration.

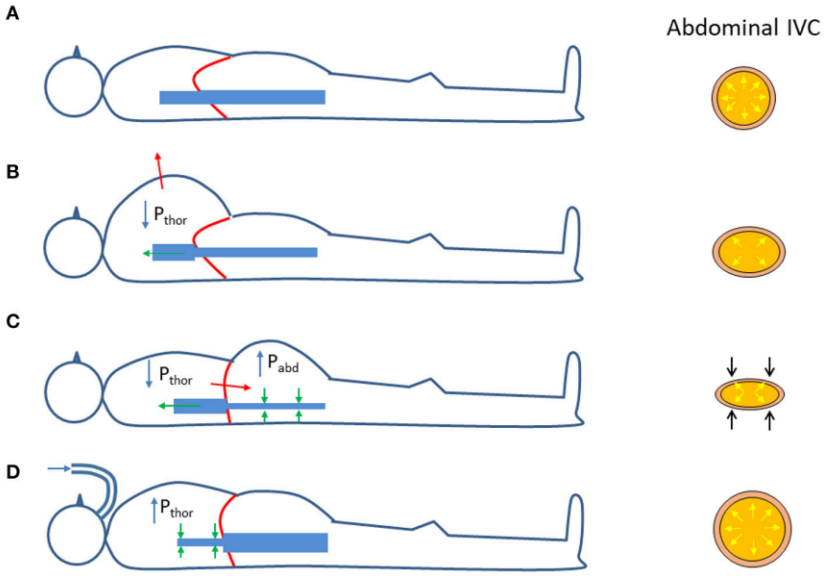


Figure 1.2: Variation of the Inferior Vena Cava based on different types of respiration. In particular, a thoracic respiration causes a decrease in pressure and volume in the IVC, while with a diaphragmatic respiration causes a higher pulsatility of the vein, compared to the thoracic one [5].

As mentioned above, an index that allows the pulsatility of IVC to be measured is the Caval Index, defined as follows:

$$CI = \frac{D_{max} - D_{min}}{D_{max}} \tag{1.2}$$

Where D_{max} and D_{min} are respectively the maximum and minimum diameter of the vein during a breathing cycle. The CI can vary between 0 and 1, where CI=0 means that the maximum and minimum diameters coincide and that therefore the IVC is completely rigid, while CI=1 means that the minimum diameter is 0 and that therefore the vein is completely collapsed: it is therefore understood that as the caval index varies, there is a conformational and dimensional variation of the vein, which can be associated with its pulsatility.

If we study the dynamics of the vein over a period of time, we see slower and faster oscillations: these are due to respiratory and cardiac stimulation respectively (figure 1.3):

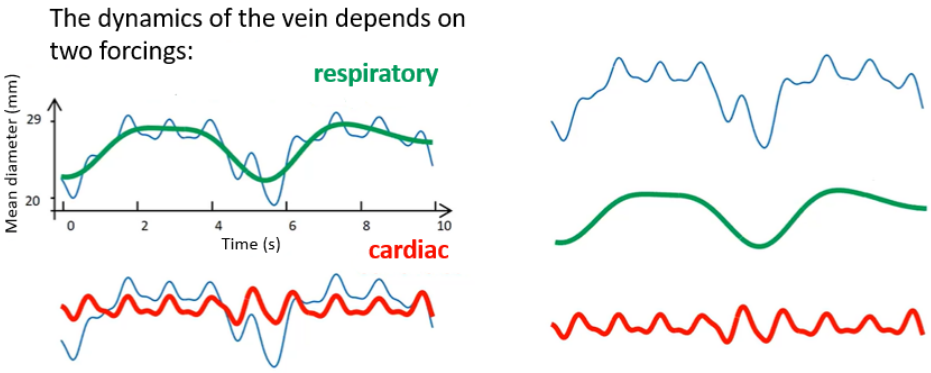


Figure 1.3: The dynamics of the IVC diameter shows both faster and slower oscillations: the former is due to cardiac stimulation, while the latter is due to the respiratory one [1].

From each of the two signals, the respective Caval Indexes can be estimated: i.e. the Respiratory Caval Index (RCI) and the Cardiac Caval Index (CCI), which can be exploited, through the construction of artificial intelligence-based models, to estimate the patient's RAP and volemic status.

1.2 Ultrasounds

Since the scope of this work of thesis is to obtain an excellent object detection model, capable of detecting the inferior vena cava within ultrasound scans, it is fair to present an overview of how ultrasounds work, meaning its

basic principles and how they are exploited to obtain meaningful images and videos.

Ultrasounds: physics and principles Ultrasounds are mechanichal waves, meaning elastic waves of compression or rarefaction, which can be schematised as a sinusoid and therefore characterized by a frequency. They are called *ultrasound* because the frequency of these mechanical waves is much higher than the human audible band which is approximately between 20 Hz and 20 kHz [8]. Ultrasounds used in medical fields are characterized by a much higher frequency: generally the ultrasound band used in medicine is in a range between 2 and 20 MHz. Since they are mechanical waves, they interact with human tissues exchanging only mechanical energy, meaning they are not ionizing radiations and therefore they can be used in such cases when other imaging techniques (such as X rays, which are ionizing radiations) cannot be used, for example during pregnancy [8].

The physical principle underlying ultrasonography is reflection. The latter is regulated by the Snell law, which states the following: considering two different and consequent mediums (characterized by two different optical coefficients n_1 and n_2 respectively), which are crossed by a beam of light with an angle of incidence equal to α, the latter will be partially reflected with the same α angle and partially refracted with an angle depending on the ratio between n_1 and n_2. The quantity of light which is in fact refracted depends on the mediums properties, such as propagation speed. In the figure 1.4 is reported a visually representation of the Snell law [8].

In ultrasonography, light is forced to cross through different mediums, but unlike optical physics, in ultrasonography is considered an orthogonal incidence ($\alpha = 0$), meaning that the light always continues in the same direction: when it finds a discontinuity, part of it is reflected (coming back to the source of the light) and the rest is refracted in the same direction [8].

In order to understand how much light is reflected and how much is refracted, we need to focus on the reflection coefficient, defined as follows [8]:

$$R = \left(\frac{Z_1 - Z_2}{Z_1 + Z_2}\right)^2 \tag{1.3}$$

while the refraction coefficient is obviously defined as:

$$T = 1 - R \tag{1.4}$$

In the equation 1.3, Z_1 and Z_2 are the acoustic impedances of the mediums crossed by the incident light. Acoustic impedance of a medium is defined

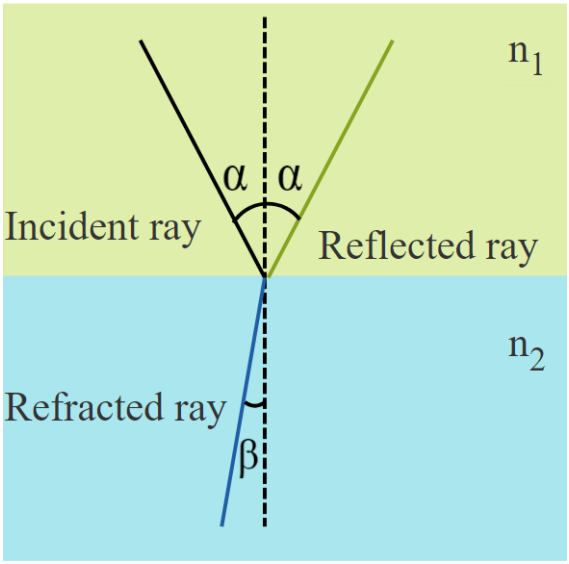


Figure 1.4: Representation of the Snell law.

as the product between its density and the propagation speed of the US crossing the same medium (the latter is assume to be equal to 1540m/s) [8]. Actually, acoustic impedance depends not only on the density, but also on the molecular structure of a tissue, in fact there are tissues with same density but different acoustic impedance and tissues with different density and same acoustic impedance. Generally, it has been seen that gases have a very low acoustic impedance (in the order of 0,0004 ray), while soft tissues (like human organs) have acoustic impedance pretty similar, ranging from 1,38-1,8 ray. On the other hand, hard tissues, like bone tissue, have a very high acoustic impedance (around 7.80 ray). This means that when US finds a discontinuity between soft tissues gas or bone, the reflection is almost total, meaning that the ray can no longer propagate through its path [8]. At this point is clear that ultrasonographic image is obtained thanks to the amplitude of the reflection echoes, coming back to the probe from the tissues crossed by the US [8].

Ultrasonographic probe In order to understand how the ultrasonographic probe works, it is reasonable to start talking about how US are generated and how they are received. US are generated exploiting the piezoelectric effect, which is typical of materials (in particular crystals) with a specific molecular structure (such as sodium chloride or some ceramics like PZT, which is actually used to fabricate ultrasonographic probes) [8]. Piezoelectric effect is a bidirectional phenomenon, which means that there are a direct and an

inverse piezoelectric effect: both of them are involved in ultrasonographic probes. With reference to figure 1.5, let's consider a cube of piezoelectric material with a certain initial length L and characterized by two metallic faces which present an electric potential V. Due to the direct piezoelectric effect, when a ΔL variation is applied to the cube, may it a compression (blue arrows) or a dilation (red arrows), a potential variation verifies between the two metallic faces: an increase or a decrease respectively with a linear dependency between length and potential. On the contrary, in the inverse piezoelectric effect, when we apply a potential variation ΔV, for example a sine wave with a certain frequency f_0, it manifests as a length variation with the same sine wave and the same frequency f_0 [8].

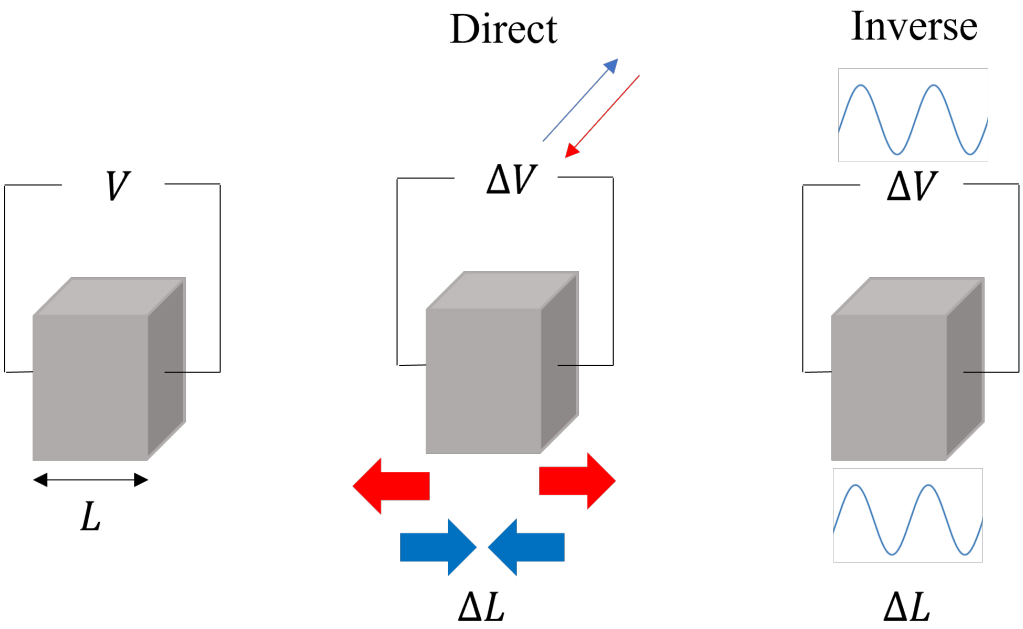
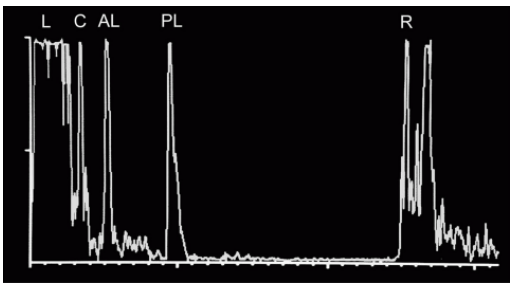


Figure 1.5: Representation of the piezoelectric effect: a variation in the physical dimension of the crystal corresponds to a linear variation of the potential that can be read between the two faces of the crystal. While a potential input (for example a sinusoidal potential) corresponds to a variation of the physical dimension (in the case of the sine input, the output will be a sine as well).

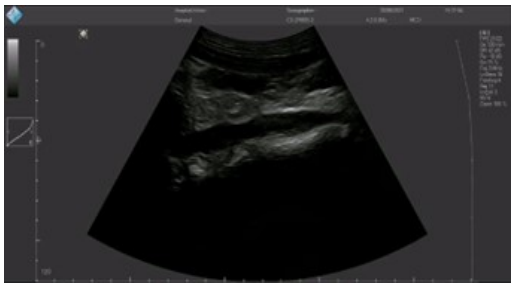
As previously said, both of these effects are exploit in ultrasonography, in fact the inverse one is used to generate ultrasound, by choosing the correct frequency. On the contrary the direct effect is used to receive the reflection echoes, in fact, since the ultrasound waves are mechanical wave, they stimulate the piezoelectric material, causing a potential variation readable from

the outside [8].

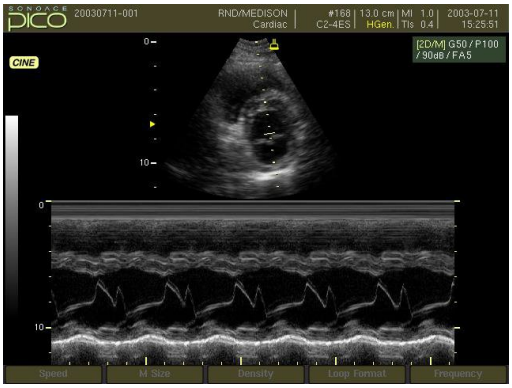
Now that it has been discussed how the image is obtained and how the US probe works, it is useful to discuss about the display modes that an US probe allows. There are three main modes: A-mode, B-mode and M-mode [8]. A-mode (which stands for Amplitude mode) is the simplest mode, but not very used today. It represents the echoes as spikes on a single axis, where the height of the spike corresponds to the amplitude of the reflected sound wave. B-mode (Brightness mode) is most used in diagnostic field. It produces a two-dimensional gray-scale image where different shades of gray represent different tissue densities: each pixel of the image has a brightness proportional to the amplitude of the feedback echo. This mode is the one used in this work of thesis. M-mode (Motion-mode) is used to monitoring moving structures. In this case, the probe investigate a single scan line and represents the movement of every interface line. This mode is usually combined with the B-mode in order to have a better visualization of the scan line [8].



(a) *A-mode*



(b) *B-mode*



(c) *M-mode*

Figure 1.6: In this figure are represented the three display modes: in figure (a) is reported an A-mode scan representing an ophthalmology exam. In figure (b) is reported a typical B-mode scan, reporting a IVC in longitudinal section. This picture is taken from the dataset used in this work. In figure (c) is reported an M-mode (lower part) coupled with a B-mode (upper part) scan, representing the mitral valve.

Chapter 2

Object Detection

Object detection is a computer vision technique that consists in locating and classifying objects in an image or in a video and labelling them with rectangular bounding boxes to show the confidences of existence.

For this reason, it finds various application in many fields of the real world, such as autonomous vehicles, surveillance systems, robotics, medical imaging, image classification, human behaviour analysis and so on, leading scientists and engineers to show an increasing interest in improving this methodology, in order to make it more and more efficient.

Object detection methods can be divided in two main branches: traditional approaches and deep-learning based methods. The pipeline of traditional object detection models can be mainly divided into three stages:

1. **Informative region selection** consists in scanning the whole image in order to find all the possible locations of the object in it (computationally expensive procedure). In particular, it mainly uses sliding-windows of different sizes and ratios to slide on the image from left to right and top to bottom by a certain step size. The image blocks cropped by the sliding window are transformed to form an image with uniform size;

2. **feature extraction** consists in extracting visual features which can provide a semantic and robust representation of the object;

3. **classification** consists in assigning a class to the object detected, using a classifier at the end of the pipeline, such as a Supported Vector Machine (SVM), and Adaboot [9].

However, traditional approaches had several flaws, for example the feature extraction stage had to be handcrafted, which means an expert operator has

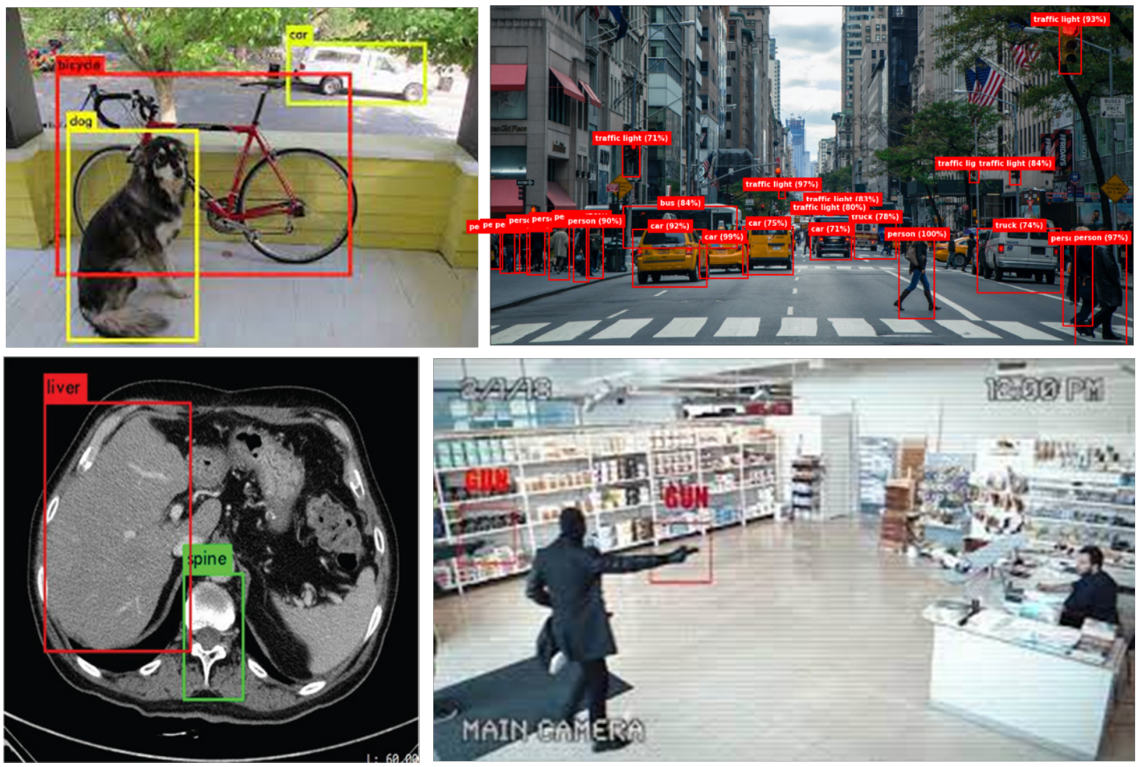


Figure 2.1: Example of object detection applications to various fields, such as medicine, surveillance and autonomous guide.

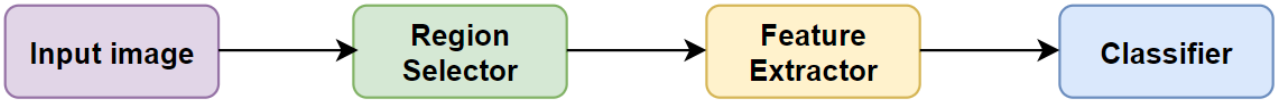


Figure 2.2: Basic architecture of a traditional model for object detection.

to "tell" the model which feature are important and which are not: this make a certain model very specific for some objects and very useless for some other. Another flaw of traditional models is the computational cost: as said before, the method consisted in multiple stages and every of them was computationally expensive, making real-time applications almost impossible. For these reasons, traditional approaches have been far outdone by deep-learning approaches, thanks to the advent of Convolutional Neural Networks (CNNs). The windward of deep learning can be attributed to the following factors:

- The emergence of large-scale annotated training data, such as ImageNet;

- Great improvements in software and hardware, like the development of high-performance parallel computing systems, such as GPU clusters;
- Significant advances in the design of network structures and training strategies, such as unsupervised learning; the use of dropout and data augmentation to solve the overfitting problem; batch normalization and so on [9].

In this way, in a few years, we were able to reach extraordinary improvements in object detection, overcoming all the limitations that characterized traditional approaches, in fact deep learning methods guarantee higher accuracy and faster performances (allowing real-time applications), moreover they don't need handcrafted feature extraction (which simplified a lot this step, in fact the network itself is capable of extracting the information it needs from the image), permitting a better adaptability to different data (that promotes the applicability of object detection to different fields in a simpler way). In the following, it will be exposed how a generical convolutional neural network works and which are the most famous networks used to perform object detection.

2.1 CNNs' Architecture

Convolutional Neural Networks (CNNs) are the most representative networks of deep learning and, as said before, the use of CNNs to perform object detection drastically revolutionized this field.

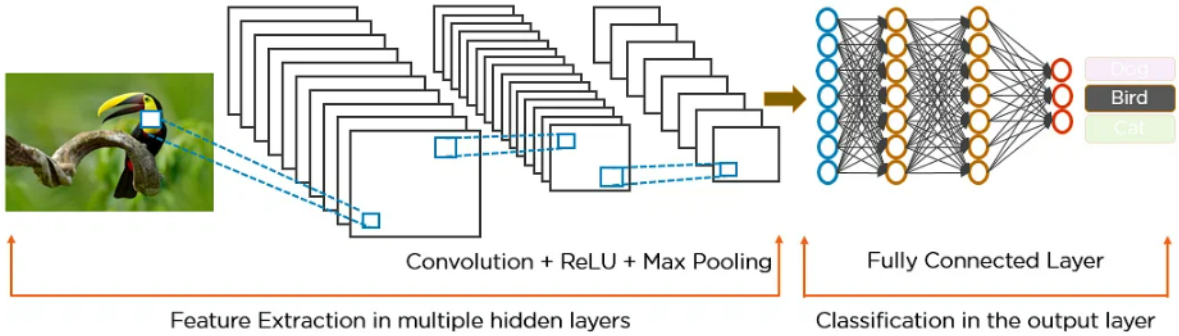


Figure 2.3: General architecture of a CNN. It can be divided in two steps: feature extraction, which is aimed to learn the most important features from the input image, and classification, whose goal is to classify the input image, based on its features.

The general architecture of a CNN (showed in figure 2.3) can be divided in two main phases: feature extraction and classification. The feature extraction phase is composed by multiple layers: convolutional layers, ReLU layers and Max Pooling layers. While the classification phase is responsible for classifying the input image.

Convolutional layers This is the first step in the process of extracting valuable features from an image. A convolution layer has several filters that perform the convolution operation between the image – considered as a matrix of pixel values – and a filter (also called kernel), composed by a squared matrix, different for every convolutional layer. The filter has the characteristic that it is smaller in space, meaning its width and height are smaller than the image, but the depth of the filter must match the depth of the image, i.e. if the image has three channels (RGB image), the filter must have three channels as well [11].

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image, known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

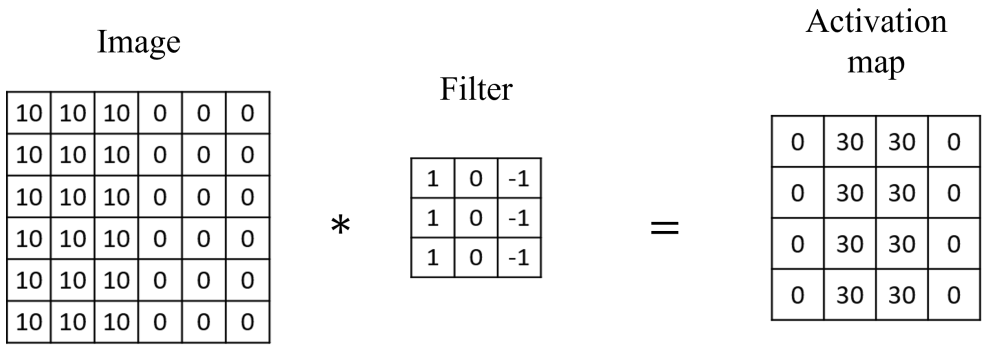


Figure 2.4: Representation of how activation maps are obtained in the convolutional layers of a CNN. An activation map is the result of a convolution between the input image and a filter (also called kernel) [11].

The dimension of the resulting activation map is obtained from the following formula [11]:

$$W_{out} = \frac{W - F + 2P}{S} + 1 \tag{2.1}$$

In which: W_{out} is the width and height of the activation map; W is the dimension of the original image; F is the spatial size of the filter; S is the stride; and P is the amount of padding (if the coupling of the kernel size with the stride is not compatible with the dimension of the image, a padding of the image is needed, which means adding a cornice of pixel with value zero to the image).

As said before, every convolution layer is characterized by a different kernel, which means that for every different convolution layer it will give a different activation map as output.

Thanks to the use of the convolutional operation, it is achieved a more efficient system, since the convolution between a matrix (i.e. the image) and a smaller filter permits the extraction of meaningful information by operating much less calculation compared to the use of fully connected layer, that instead would require lots of calculation – and so, a higher computational cost – since images can have millions or thousands of pixels. Convolutional layers permit to store fewer parameters, which means that not only the memory requirement of the model is reduced but also the statistical efficiency of the model is improved [11].

ReLU layers After the first convolutional layers, it is applied the ReLU (Rectified Linear Unit) activation function to the activation maps outgoing from the first layers, generating as output a rectified feature map. The ReLU function is described by the following formula:

$$\varphi(z) = max(0, z) \tag{2.2}$$

Which means that it compares the input z with 0: if z is greater than 0, then it passes unaltered, conversely, if z is smaller than 0, then it is converted to 0. This mechanism is shown in the graph in figure 2.5.

This operation introduces non-linearity to the network, which is crucial for enabling the network to learn complex relationships and features in the data, moreover, this mechanism alleviates the vanishing gradient problem that can occur during training deep neural networks.

Pooling Layer After the ReLU layer, the activation maps need to be processed by a pooling layer. The pooling layer replaces the output of the network so far (i.e. the activation maps) at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and

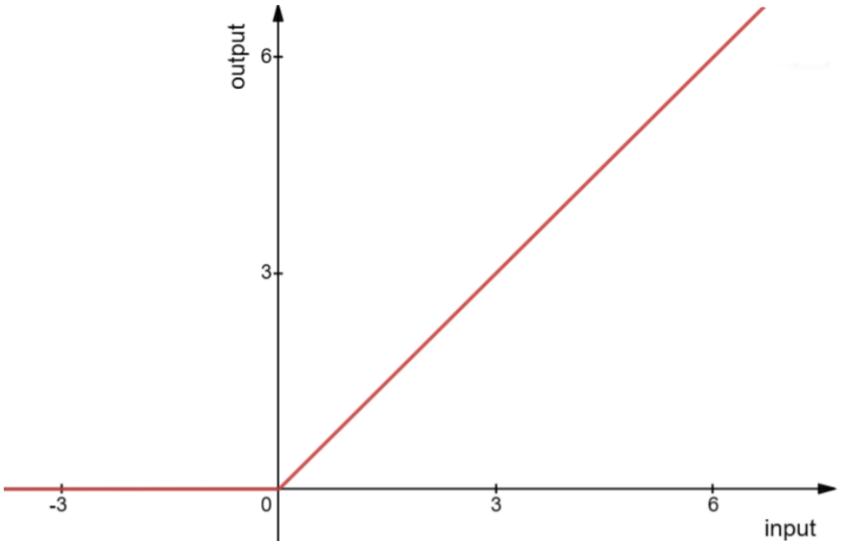


Figure 2.5: The ReLU function makes sure to set to zero all the inputs lower than zero, while it leaves unchanged all the input higher than zero.

weights. The pooling operation is processed on every slice of the representation individually [11]. An example of how pooling layers operate is reported in figure 2.6.

There are several pooling functions such as the average of the *rectangular neighbourhood, L2 norm of the rectangular neighbourhood* and a *weighted average* based on the distance from the central pixel. However, the most popular process is *max pooling* , which reports the maximum output from the neighbourhood [11].

This operation helps the network identifying some specific parts of the image, in addition reducing the size of the image, the computational time reduces too. Just like in the convolutional layers, the pooling layer is characterized by filters, that could come in different sizes, and that control the size of the sub-squares. The size of the activation map, deriving from the pooling layer, is dictated by the following formula [11]:

$$W_{out} = \frac{W - F}{S} + 1 \tag{2.3}$$

In which W is the original size of the activation map; F is the size of the filter of the pooling layer; S is the stride.

After the pooling layer, the new activation maps are flattened, it means that they are transformed: from matrices they become column vectors. After this passage, they are fed into a fully connected layer.

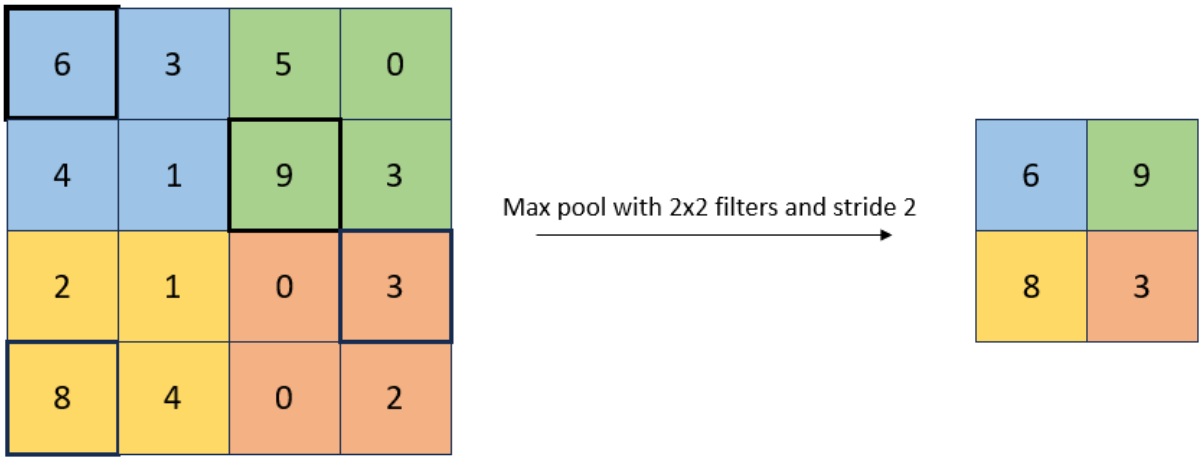


Figure 2.6: Example showing how max pooling works. For every sub-square of dimensions $4x4$, it is maintained only the highest number.

Fully connected layer This is the last step of the convolutional neural network pipeline: it is the stage that performs the classification of the image it was fed with. The fully connected layer contains neurons that connect to the entire input volume, as in ordinary neural networks. The last layer of neurons has as many neurons as many are the classes to be identified.

2.2 Networks for object detection

As mentioned above, the main networks used for performing object detection are based on CNNs, since the traditional methods didn't lead to satisfying results and, in addition, they were characterized by several flaws that did not allow an efficient functioning. For this reason, the current state of the art of the neural networks used for object detection is composed completely by networks based on convolutional neural networks. This kind of networks can perform detection in multi-class datasets, in images with occlusions (meaning the target is occluded by some other object, like clouds, shadows and so on), where the background sequence is changing and in large datasets [10]. On the other hand, these networks have some drawbacks, among these we have that they are computationally more expensive, compared to the traditional methods, they are prone to localization errors due to fast speed and specialized systems based on GPUs are required to train and evaluate these methods [10]. Some of these networks are: Region Proposal based Convolutional Network (R-CNN) and the family around this such as Fast-RCNN,

Faster-RCNN or Mask-RCNN; Single Shot Detector; Spatial Pooling Pyramid Networks; and YOLO family. In this section, they will be exposed some of the main networks previously mentioned, how they work, their strengths and their weaknesses.

Region Proposal Convolutional Networks (RCNNs) This method was proposed by Girshick et al. in 2014 [12]. Unlike the traditional methods, that involved the analysis of every region of the image (resulting in a very time-consuming operation), region proposals methods consist in looking for selective regions in the image to locate the object. The pipeline the characterizes these methods can be divided in two step: in a first step the selective regions are found by a selective search method; the second step perform the actual object detection, using a convolutional neural network that analyzes every region proposed by the first step [10].

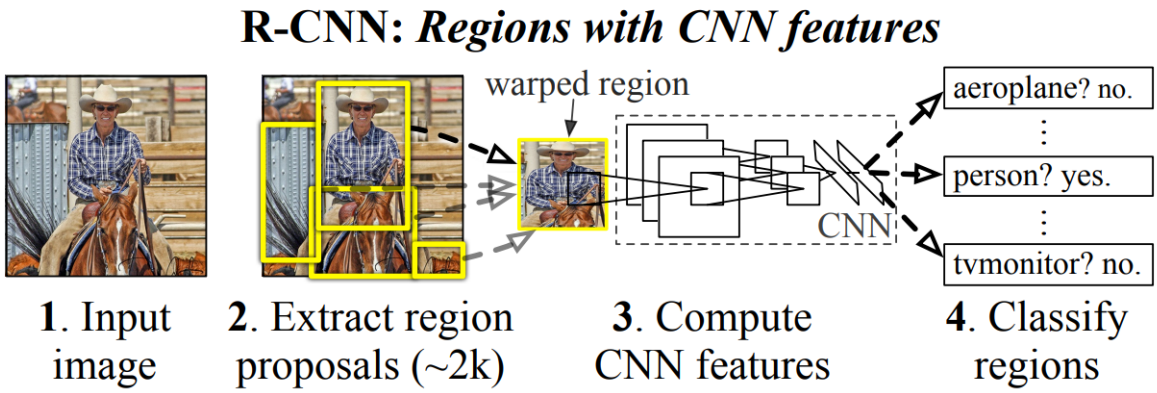


Figure 2.7: Overview of the RCNN system. (1) This system takes an input image, (2) extracts around 2000 region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs [12].

The pipeline of the RCNN can be summarized as follows [10]:

1. Selective search:
   (a) Multiple regions are generated after applying a segmentation to the input image, which can have an arbitrary size;
   (b) To reduce the number of regions, several small regions are aggregated basing on colour, texture, size similarity and shape compatibility;

(c) Regions of interest are identified among the large regions obtained in the previous step. In these regions of interest, object detection is performed.

2. Object detection:

   (a) A pre-trained convolutional neural network model is re-trained such as the last layer has as many neurons as the number of classes that are to be detected;

   (b) For each image, regions of interest are collected and reshaped to fit into the CNN model input;

   (c) A Support Vector Machine (SVM) based classifier is trained to classify the image into object and background. A binary SVM is trained for each class.

   (d) In this step, tight bounding box is applied across the images. This is performed by training a linear regression that classifies each category of object in the image.

This method allows also the detection of the background objects in the image, furthermore it is less prone to localization errors, thanks to the region proposal method. However, it is characterized by several flaws, including the fact that is computational very expensive, since for each image are extracted 2000 region proposal, resulting in $Nx2000$ features (where N is the total number of images) to be calculated. Furthermore, it is a very slow model that doesn't allow real-time applications [10]. The latter is the main reason why this model was excluded for the purpose of this work.

Mask RCNN This method, proposed by He et al. in 2017 [13], is an extension to Faster-RCNN and performs prediction for object mask, in particular, it adds a branch in parallel with an existing bounding box recognition branch for predicting an object mask [14].

Also in this case, the pipeline of the Mask RCNN is composed by two step: the first step performs object detection, through Faster RCNN: in this step classes and bounding boxes are obtained; the second step performs semantic segmentation through a fully convolutional network: the latter is applied to classes and bounding boxes outing from the first step, in order to obtain pixel wise boundary of the object classes.

This method allows detection at pixel level boundary of object classes, permitting more accurate bounding boxes. On the other hand, it is a slow

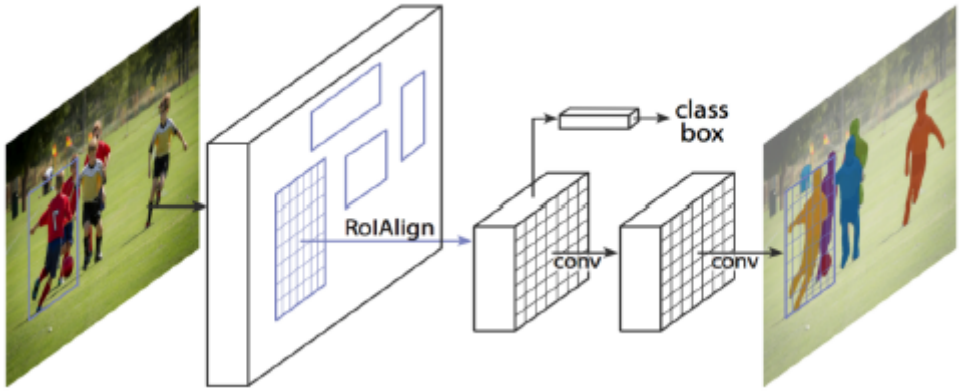


Figure 2.8: Schematic pipeline of Mask RCNN functioning.

method, due to the use of two parallel methods, this makes it not suitable for real-time applications.

Single Shot Detector Single shot detector was proposed by Liu et all. in 2016 [15]. This method discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes [15].

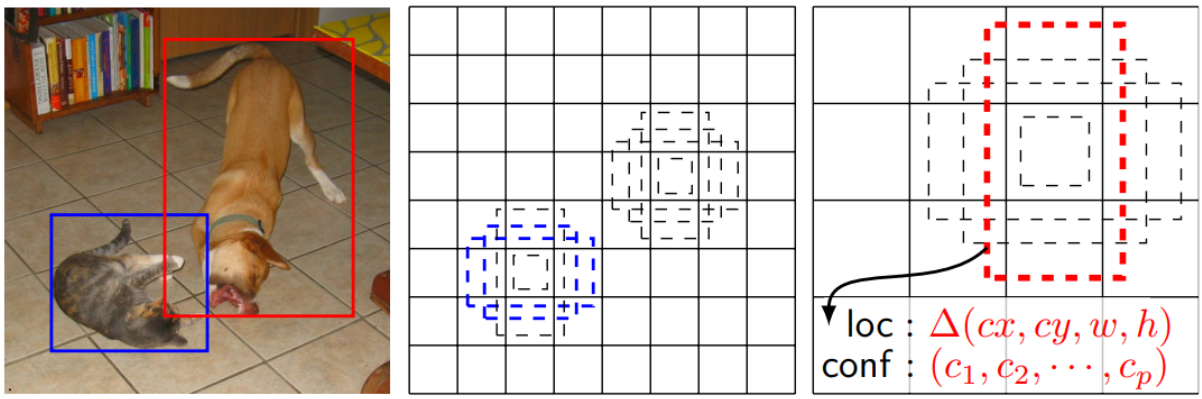


Figure 2.9: General idea of the Single Shot Detector framework.

For what concerns specifically the method used by Single Shot Detector model, it can be divided into three steps: in the first place a convolutional layer is used to predict object classes, by passing the input image from a fixed sized set of bounding boxes and the object class presence in a bounding box is measured using bounding box scores. In a second step, it is necessary to filter the multiple bounding boxes around the same object: to do so, a non-max suppression is applied. The latter makes sure to keep bounding boxes with higher scores and to hide bounding boxes with lower scores. In the final step, in order to compare predictions and ground truth IoU intersection is applied, during the training time.

This method has several advantages and disadvantages [10]. The advantages include the high speed and accurate detection and also the ability to perform detection in multi-resolution images. On the other hand, among the disadvantages we find the fact that the training time is very slow, because this method is based on a VGG-16 network; in addition it confuses object belonging to the same class and it has some difficulties in detecting small objects.

YOLO The first YOLO model was proposed by Redmon et. all in 2016 [16] and, throughout the years, several different YOLO models have been released, such as YOLOv3, YOLOv4, YOLOv5, and so on, until the latest model, which is YOLOv8, released in 2023. In this paragraph, the main characteristics of the first YOLO model are described.

Differently from previous models, YOLO has the advantage of being a single stage detector: meaning that while until now every model was characterized by two stages performing object detection (for example RCNN family), in this case a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance [16]. This innovation permitted to drastically reduce training and evaluation time, allowing also real-time applications.

For what concerns the training pipeline and the how the model works, it is briefly illustrated in figure 2.10 and explained right after.

The pipeline followed by the model is the following [10]:

1. The input image is divided into an SxS grid. Each grid is a cell responsible to predict the object centred in that grid cell.

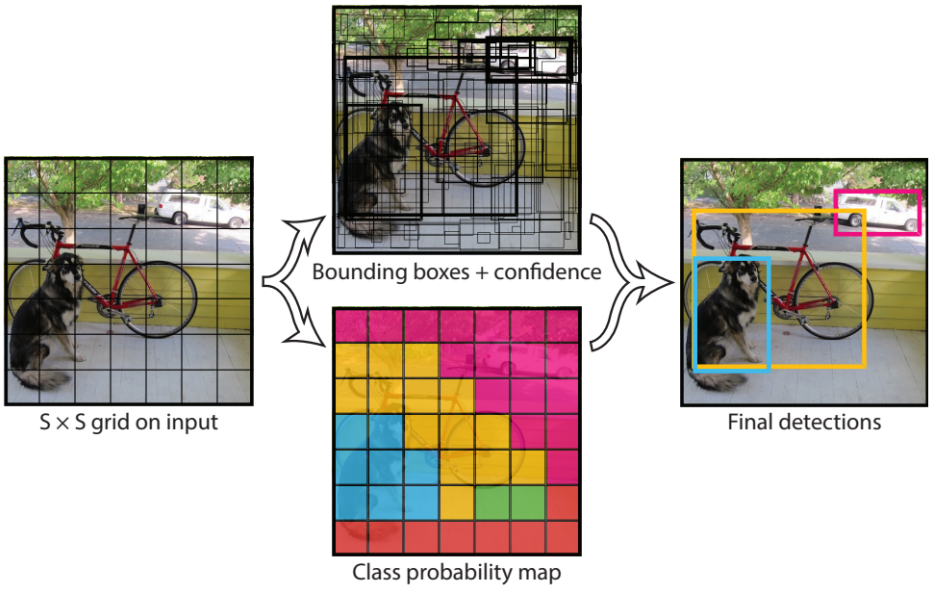


Figure 2.10: Illustration of how the first YOLO model works [16].

2. Each grid predicts B bounding boxes and their confidence scores. Confidence scores are defined as

$$Pr(Object) * IOU^{\text{truth pred}} \tag{2.4}$$

$Pr(Object)$ is the probability of finding the object in a certain grid, while $IOU^{\text{truth pred}}$ stands for Intersection Over Union, where the intersection and the union are between the ground truth and the prediction of the model. It is clear that IOU is at its best when it is nearer to 1, since intersection and union are similar, while if it diverges, it means that the intersection is near to 0, meaning a bad prediction.

3. Simultaneously to step 2, for each grid cell and regardless of the number of boxes, Conditional Class probability C is also predicted as

$$Pr(Class_i|Object) \tag{2.5}$$

These probabilities are conditioned on the grid cell containing an object.

4. At test time, Conditional Class probabilities and the individual box confidence predictions are multiplied:

$$Pr(Class_i|Object) * Pr(Object) * IOU^{\text{truth pred}} \tag{2.6}$$

which gives class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

The architecture of the first YOLO network is based on DarkNet model and it is composed of 24 convolutional layers, followed by 2 fully connected layers. The complete architecture is shown in fig 2.11.

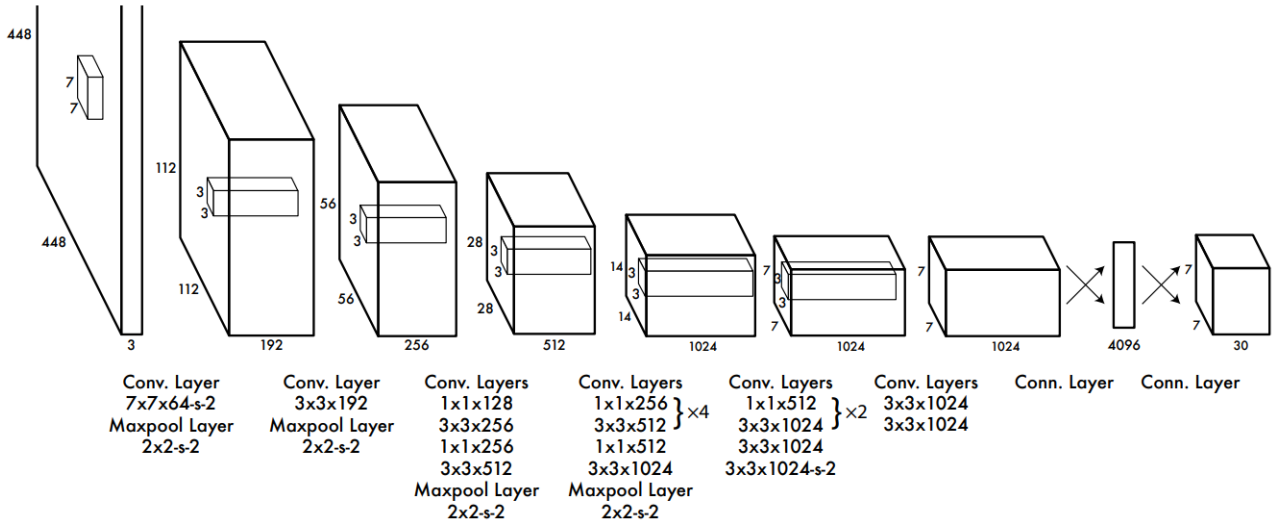


Figure 2.11: Architecture of the first YOLO network [16].

This network has brought many advantages compared to two stages detectors, used until then. Among these advantages we found it is very fast, since it can reach detection at 45fps: this characteristic allows real-time applications. Moreover, the developing of smaller architecture, allowed faster versions of YOLO that can perform detection up to 155fps. Another advantage brought by the usage of YOLO is that it can generalize very well [10], meaning that it is able to achieve good performance with a large variety of data. On the other hand, it has some limitation too, for example it is prone to localization errors, due to the fact that the loss function is designed to treat errors in the same way in small bounding boxes and in large bounding boxes. Moreover, this network struggles in detection of small objects, since each grid cell only predict two boxes and can only have one class: this limits the number of nearby objects that the model can predict [16]. However, despite the limitations that YOLO has, the choice for this work of thesis fell on this family of networks, thanks to its superior advantages compared to the other networks analyzed. In particular, it has been chosen to use the latest

network that was released, meaning the YOLOv8, releases, as said before, in 2023. In the next chapter it will be analyzed in detail its architecture and the design choices that led this network to be the current state of the art in the object detection field.

Chapter 3

YOLOv8

3.1 YOLOv8's Architecture

As mentioned above, for this work of thesis YOLOv8 has been chosen to perform object detection. In order to precisely understand how it performs detections is useful to understand its architecture: in this chapter it will be exposed YOLOv8 architecture.

Before starting it's important noting that currently Ultralytics group has not released an official paper of YOLOv8 yet, in which are described the main feature of the model, its architecture and so on. This is due to the fact that they are focused on improving the network in all its functionalities, but the developer group has expressed their intentions in releasing an official paper in the near future. As a result of the lacking official paper, YOLOv8 remains relatively uncharted territory within academic circles, with no official publications or in-depth explanations of the model's internal workings available from Ultralytics, the creators of YOLOv8. Despite the absence of an official publication, the model has garnered significant attention and interest from the wider community. This has led to active engagement between the community and the Ultralytics team, as they seek to gain a better understanding of YOLOv8, underscoring its potential usefulness. Given that, in order to provide a comprehensive description of YOLOv8's main features and working, it has been conducted an in-depth study of the YOLOv8 model's repository [17]. In fig 3.1 the detailed architecture of YOLOv8 is shown. As can be seen in the picture, the architecture can be divided into two parts: the backbone and the head.

YOLOv8's architecture is based on YOLOv5's architecture, but the authors have brought various changes in terms of model scaling and architecture

tweaks.

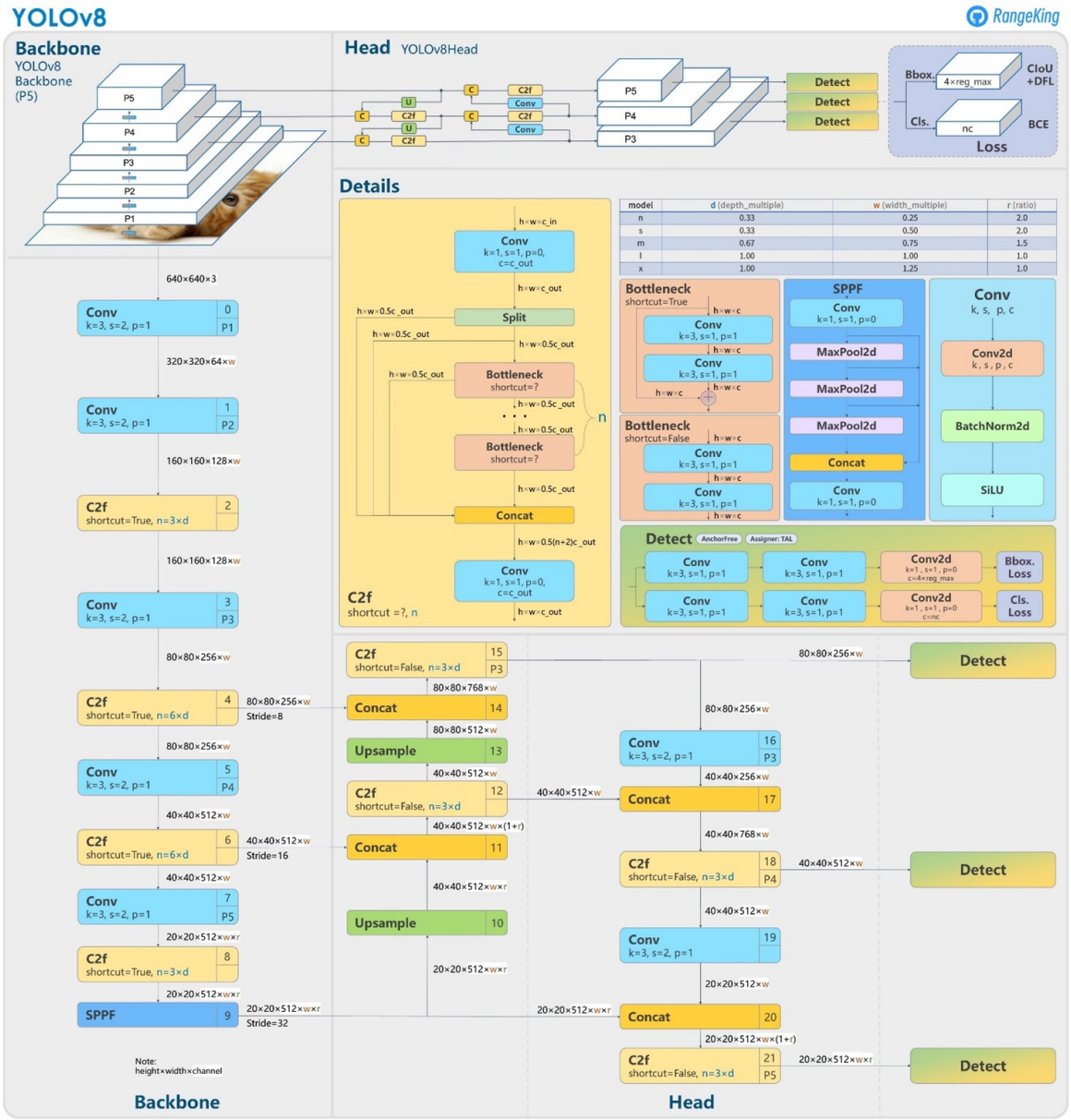


Figure 3.1: YOLOv8 architecture [18].

Backbone In this case, the backbone used by YOLOv8 is the CSPDarkNet53, which is a DarkNet53 network improved with an approach called CSP (Cross Stage Partial Network). DarkNet53 was developed by Joseph Redmon,

the creator of the first YOLO model and it was specifically designed to perform object detection.

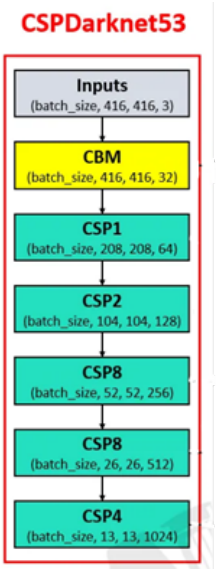


Figure 3.2: Simplified architecture of DarkNET53.

Since it is used as a backbone, it acts as a feature extractor. DarkNet53 is basically a series of convolutional layers, in particular 53 layers, as the name suggests, that have the goal of extracting relevant features from the input image. Each convolutional layer is followed by a batch normalization (BN) layer and a SiLU layer, as activation function. The Sigmoid Linear Unit (SiLU) function has the following mathematical expression:

$$silu(x) = x * \sigma(x) \tag{3.1}$$

Where $\sigma(x)$ is the logistic function. As can be seen in fig 3.3, this function outputs only inputs that are greater than zero, and outputs them equal to themselves.

As can be seen in fig 3.1 after each convolutional layer, the channel output dimension is doubled, going from 64, then 128 and so on, until the last convolutional layer of the backbone that is characterized by a channel output dimension of 512: this is done in order to increase the complexity of feature representations of YOLOv8 network: this allows the latter to detect more complex features and, consequently, to generate more accurate detections.

The CSP modification was introduced in the YOLO family with YOLOv4 and it is represented in the C2f module (cross-stage partial bottleneck with 2

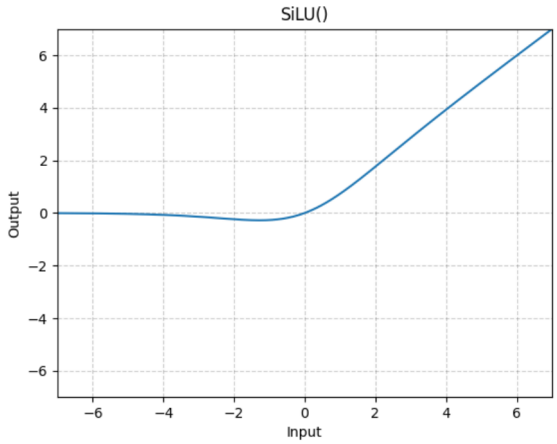


Figure 3.3: SiLU activation function.

convolutions). It is based on the CSPNet strategy, which implies the splitting (split block in the C2f module) in two parts of the feature maps of the base layer and the subsequent merging of the same through the cross-stage hierarchy: this strategy provides a reduction of calculation while keeping good accuracy [19].

Another strategy used by YOLOv8 is the usage of Feature Pyramid Networks (FPN) [20], which is a specific architectural design pattern that can be incorporated into the backbone of an object detection model to further enhance its multi-scale feature extraction capabilities. In fact, fusing features from different layers can improve their expressive ability.

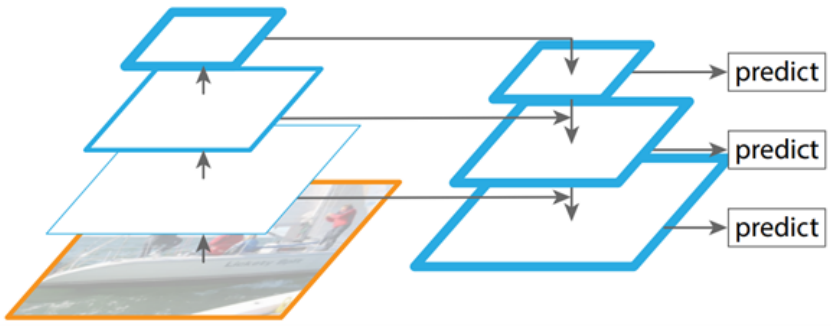


Figure 3.4: Feature Pyramid Networks (FPN) structure [20]. It is composed by two pathway: a bottom-up and a top-down one, moreover lateral connections make the two pathway communicate.

As can be seen in figure above, FPN composes of a bottom-up, a top-down pathway and lateral connections. The bottom-up pathway is the normal feed-forward computation of the backbone that perform feature extraction. This pathway computes a feature hierarchy in which the feature maps are progressively downsampled – with a scaling step of two, meaning that after each convolutional layer the feature maps are half size of the previous map – in order to obtain smaller feature maps but more significant. So we can state that as we go up, the spatial resolution decreases, while with more high-level structures detected, the semantic value for each layer increases.

The top-down pathway constructs higher resolution layers by upsampling spatially coarser, but semantically stronger, feature maps from higher pyramid levels. In addition, through the lateral connections, these features are enhanced by taking into account the feature maps of the same spatial size from the bottom-up pathway. This improvement allows a better understanding of the locations, which can be difficult because of the downsampling and upsampling processes [20]. In this way, the feature maps coming out of the top-down pathway are semantically strong and more accurately localized, thanks to the lateral connections. Looking at the architecture of YOLOv8, the lateral connections are located between the C2f module and the concat module belonging to the top-down pathway.

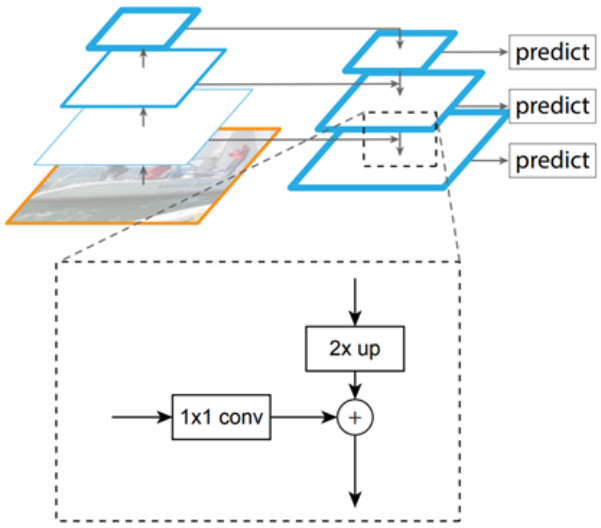


Figure 3.5: Building block a the top-down pathway [20].

In figure 3.5, it is represented the building block of the top-down. As we can see, the entity of the upsampling operated each time is by a factor of two. Then, the upsampled map is merged with the bottom-up feature maps,

which is got through a 1x1 convolution.

Head The head is responsible for taking the feature maps generated by the backbone and further processing them to produce the final output of the model in the form of bounding boxes and object classes. The head of YOLOv8 (and in general of the latest YOLO models) is designed as a Path Aggregation Network (PAN). The general topology of the PAN is represented in the figure below.

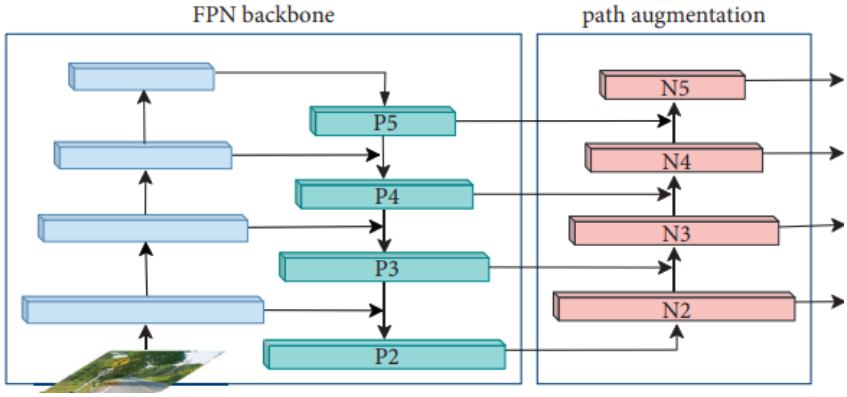


Figure 3.6: Path Aggregation Network (PAN) topology.

Despite the lateral connections provided by the FPN structure of the backbone, the deep features captured by the latter lack detailed information, and simple upsampling operations cannot recover the details very well. For this reason, in the head it was introduced the PAN topology: the latter makes sure to provide a supplement of detailed information for these deep features by propagating the response of shallow features coming from the top-down pathway of the FPN. Practically, the feature maps are first downsampled with a convolution with stride 2 (so that they will be dimensionally consistent) and then combined with the feature maps coming from the top-down pathway through an element-wise addition. In this way the final feature maps are characterized by both detailed information and the semantic information of the deep features, which can improve the positioning accuracy and classification accuracy [21].

Finally, as we can see in fig 2.11 the head of YOLOv8 is designed to be decoupled, meaning that it processes objectness, classification, and regression tasks independently. This design allows each branch to focus on its respective task and improves the overall accuracy of the model. To process the feature

maps, the head uses a series of convolutional layers, followed by a linear layer to predict the bounding boxes and class probabilities.

Another strategy used by YOLOv8 and firstly introduced by YOLOv4 that aims to reduce the complexity of the network while maintaining high the detection accuracy, is the anchor-free approach. The detection process of the anchor-based models starts with a series of default bounding boxes of different sizes and aspect ratios, from which an offset is calculated as a candidate box; then a loss is computed based on the ground truth examples and finally it is calculated a probability that a given offset box overlaps with a real object: if that probability is greater than a certain threshold the bounding box is considered valuable. Basically, an anchor-based model relies on anchor boxes to guide them in predicting the location of objects within an image. Conversely, anchor-free models like YOLOv8 don't rely on predefined boxes, but instead they directly predict bounding boxes and class probabilities from the features extracted from the input image. Thanks to this new approach, the network loses complexity and gains flexibility, allowing a better detection of objects of different sizes and shapes, particularly thin or small objects (which may not fit well the default anchor boxes). In addition, the model is much simpler compared to the anchor-based models.

3.2 Training

Before going further into the description of the work performed, it is fundamental to expose how the dataset was obtained and what operations have been performed in order to make it feasible for training the YOLOv8 model.

Dataset

The dataset was provided by Professor Mesin, and consists of ultrasound scans obtained over several years, between 2017 and 2023. The ultrasound videos are subdivided into videos representing the inferior vena cava both in longitudinal and transverse views: these are the two classes with which the YOLOv8 network was trained on.

In order to augment the variability of the dataset, the scans were taken from different sources, in particular from different ultrasound machines, different operators and, of course, different subjects, so that to have different conditions in terms of anatomical structures, image quality and so on. Furthermore, among the different subjects, pathological ones were included too,

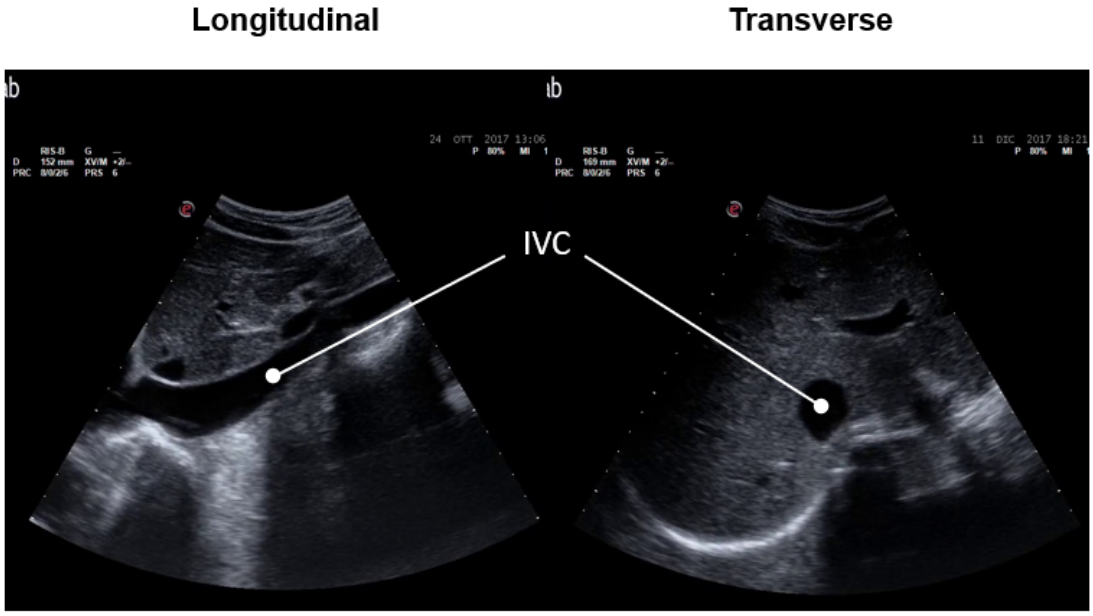


Figure 3.7: Examples of IVC scans both in longitudinal (left picture) and transversal (right picture) scans.

for example some of the pathologies that were present were: pulmonary interstitial disease, idiopathic pulmonary fibrosis, lung cancer, chronic obstructive failure, and so on. All of these are respiratory pathologies that can also affect the inferior vena cava behaviour. In this way the algorithm will be robust in a larger number of detectable cases.

In the graph 3.8 it is shown the number of scans that were processed to obtain the effective dataset to train YOLOv8 on. In particular these 127 videos were characterized by various differences, in terms of number of frame rate (which can vary from 15FPS to 50FPS) and in terms of duration of videos, ranging from a few seconds to 15-20 seconds. Keeping in mind these information, it was obtained a dataset composed of 38559 frames. Being this number more than sufficient to train the network and being the close frames very similar, it was chosen not to use them all, but to select, in the scans with the biggest number of frames (which also showed the highest frame rate), 1 frame over 5, so that to reduce overfitting and augment the variability of the dataset: in this way it was obtained a final dataset of 15960 frames.

For what concerns the dimensions of the scans, they are shown in the graph 3.9.

As can be seen in the graph, three were the predominant dimensions that characterized the dataset used: 1068x800, 1200x932 and 1172x608. Due to

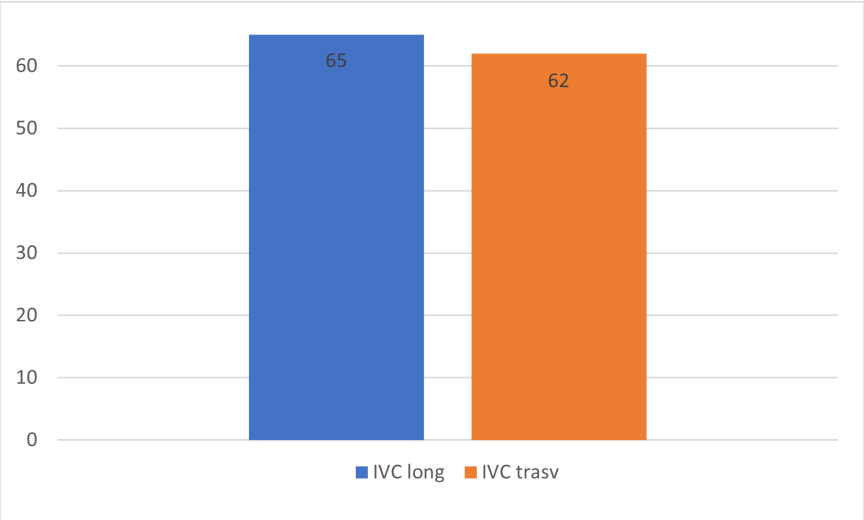


Figure 3.8: Number of IVC scans processed.

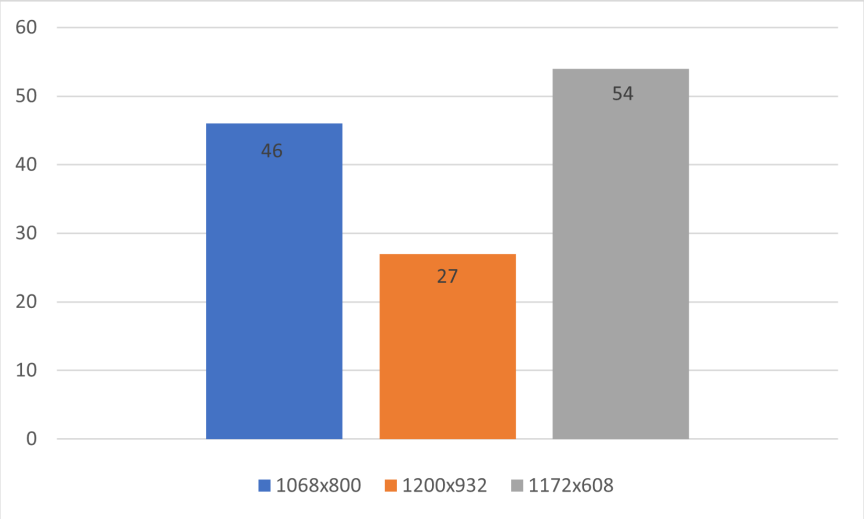


Figure 3.9: Histogram showing the different dimensions characterizing the dataset.

the fact that YOLOv8 is currently only trainable with images characterized by an aspect ratio of 1, i.e. with square images, whereas the available dataset was characterized by frames with an aspect ratio greater than 1, it had to be pre-processed. Therefore, in order to obtain square images, it was chosen to use the dimension of 800x800, mainly due to the fact that the training of a YOLO model - including YOLOv8 - requires images with a size that must be a multiple of 32, because the architecture divides the input image into a grid structure, and the size of the grid cells impacts the accuracy of the

detections. With this purpose, it was decided not to perform a simple frame resize, as it would not have been possible to maintain the original aspect ratio, inevitably leading to a distortion of the image. The images with dimension 1068x800 were treated differently from the other two categories: in fact, in this case a simple center crop of the image was performed in order to have frames of size 800x800, which obviously contained the useful information of the ultrasonographic video. While for the other two categories, it was chosen to operate a resize that transforms the biggest dimension into 800 and the other one is modified as consequence, in a way that did not change the original aspect ratio. To compensate the rectangular derived image, it was carried out a letterboxing of the latter, so that to add black bands (which don't bring any meaningful information) to the top and the bottom of the image: in this way square images were obtained.

In order to train a generic object detection model, it is necessary to provide the training, validation and test sets images with the corresponding bounding boxes, which indicate, through a rectangular box, where on the image the object to be detected is located. The available dataset was not a labelled dataset, so it was necessary to provide a reference bounding box for each frame. This was done using an open-access online software, namely CVAT [28]. What makes CVAT [28] a suitable tool for the annotation task is its interpolation capability, allowing it to automatically compute linear changes in the size and position of bounding boxes between consecutive frames. This feature facilitates and accelerates the annotation process, by removing the necessity of manually adjusting bounding boxes for each frame. At the end of this process, an example of the obtained images is shown in figure 3.10.

The annotation coming out from the software CVAT must be in the YOLO format, so that the YOLOv8 network can process the images in the correct way. The YOLO format is described as follows:

<class> <x center> <y center> <width> <height>

in which the class is the class of the object inside the bounding box and the other parameters are the coordinates of the bounding box. The latter must be normalized with respect to the dimension of the image, meaning that the values must range from 0 to 1. For each frame is generated an annotation of this kind, in which, in general, each line refers to a single object (in the case of this thesis, each ground truth annotation has one single line, because each frame has one single IVC).

For what concerns the subdivision in training, test and validation sets, it

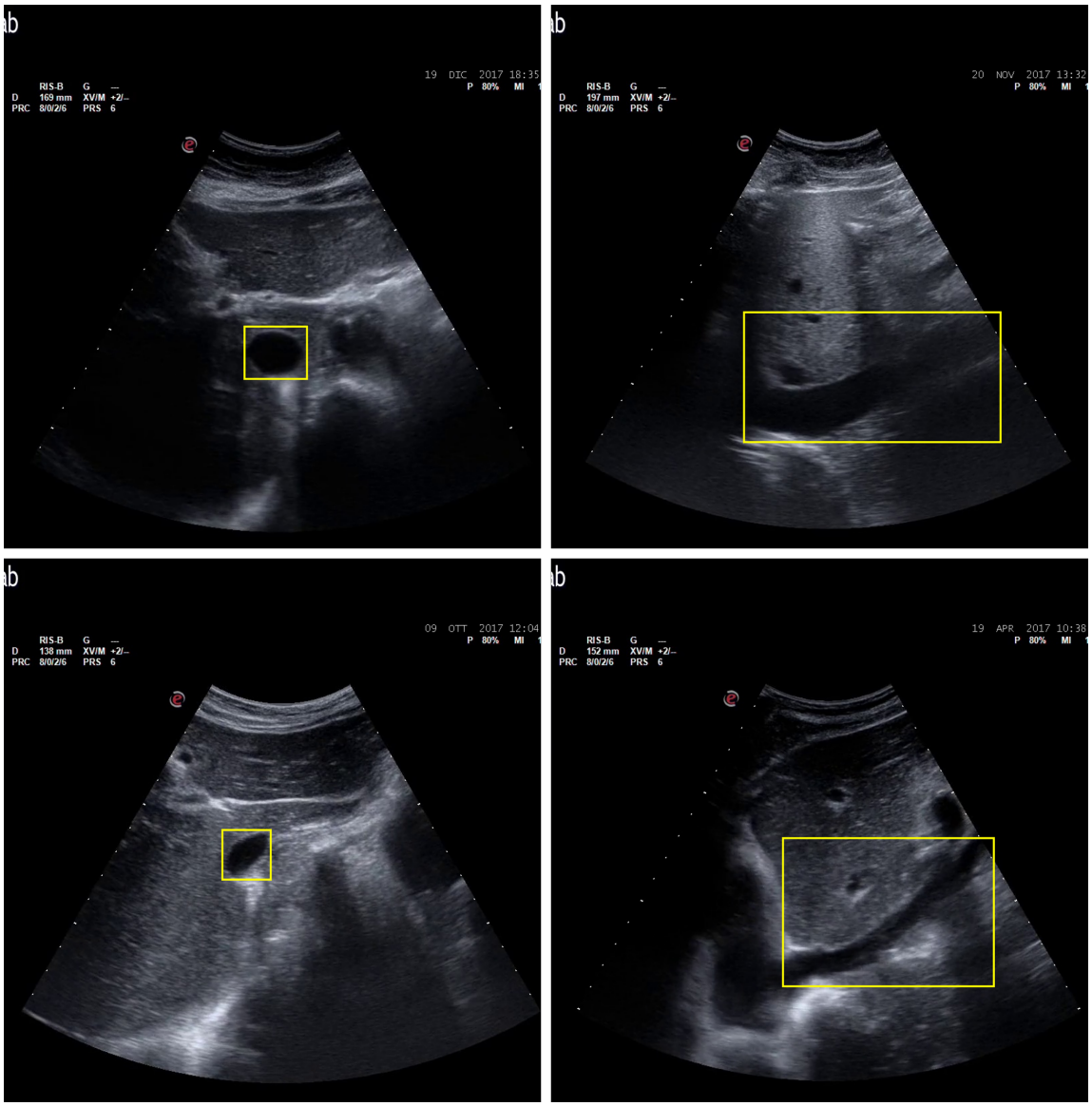


Figure 3.10: Examples of labeled frames from the dataset. In particular the images on the left show bounding boxes for transversal IVC, while images on the right show bounding boxes for longitudinal IVC.

was decided to use the 70% of the total disposable images for the training set, 20% for the validation set and the remaining 10% for the test set. In this way the training set was composed of 11.151 images, the validation set has 3.186 images, while the test set 1.593 images. It's worth noting that in the dataset were also present frames in which the inferior vena cava was not visible. In particular there were 276 background images in the training

set and 67 in the validation set. It was decided to kept those frames in the dataset: this choice was made out of the fact that in this way YOLOv8 learns not only to recognize the IVC, but also to understand where there is no IVC to detect. The presence of the background class is very important for the model training: it helps to improve the overall performance.

Training

After exposing how the dataset was obtained and elaborated, it is the moment to discuss about the training carried forward in this thesis. First of all, it is worth saying that the training and the inference phases were operated using Google Colaboratory (Colab in short) Pro, which is a cloud-based service developed by Google that provides users with a collaborative platform for coding and data analysis. In particular, it was used the Google Colab Pro plan, because it supplies more rapid GPUs and bigger RAM, compared to the basic plan. In the table below are reported the technical details of the hardware used.

Table 3.1: In this table is reported the hardware used for the training and the inference of the models.

	GPU	**RAM**
Training	A100	40GB
Inference	T4	16GB

Before continuing, a premise is in order. The Ultralytics group have made of YOLOv8 a very easily scalable model, in fact they have made use of scaling factors for the number of layers (depth) and the number of filters (width) of the network: in this way, five different models have been made available, which are called *nano, small, medium, large* and *extra large*. As the names suggest, the *nano* model is the smallest in terms of both depth and width and this makes of it the fastest model but the less accurate. On the contrary, the *extra large* one is the biggest model and, for this reason, is the slowest but the most accurate model. Given this variety of models, thus of choice, one can use the model the best suits their dataset and their purposes. In the case of this work of thesis, what is most important is that the model can work in real-time, so it was decided to prefer speed over accuracy and, for this reason and for the purposes of this work, the *nano* model was chosen. In table 3.2, it is reported a summary of the combination of the parameters

that distinguish the different models and, in green, is highlighted the chosen model.

Table 3.2: Different available models for YOLOv8. In green is reported the chosen model.

model	**d** (depth multiple)	**w** (width multiple)
n	0.33	0.25
s	0.33	0.50
m	0.67	0.75
l	1.00	1.00
x	1.00	1.25

In order to obtain the best performances possible from the training, a fine-tuning of the hyperparameter has been done, meaning different attempts have been carried on, trying different combination of the values of the hyperparameters. At the end, the hyperparameters that showed the best performances with this dataset are shown in table 3.3.

Table 3.3: Hyperparameters chosen for the training of the YOLOv8 network.

Hyperparameter	**Value**	**Purpose**
epochs	70	Number of epochs to train for.
imgsz	800	Size of input images.
batch	64	Number of images per batch.
lr0	0.001667	Initial learning rate.
lrf	0.001	Final learning rate ($lr0 * lrf$).
warmup_epochs	3	Number of epochs in which the learning rate increase, before starting decreasing.
optimizer	AdamW	Kind of optimizer to use during training.
cos_lr	True	Whether to use a cosine schedule for the learning rate during train.
close_mosaic	20	Number of final epochs, for which to disable mosaic augmentation.

In particular, for what concerns the number of epochs, it has been chosen to use 70 epochs, in fact, when choosing a higher number of epochs, for example 100, the network showed overfitting, meaning that it performed very well on the known data of the training set, but it was not able to

generalize anymore: its performances went down in cases of unknown data. The *warmup_epochs* parameter refers to the number of epochs during which the learning rate increases until the value set as hyperparameter is reached. This technique is useful for the stability of the model in the first stages of the training. As optimizer, it has been chosen the AdamW, which is a variant of the Adam optimizer. In particular, it has been included the weight decay directly into the Adam optimizer. Weight decay is a regularization term that penalizes large weights during training to prevent overfitting. For what concerns *cos_lr*, it is the schedule of the learning rate during train. In particular, when it is set to "False", the schedule involves the learning rate decreasing linearly with the epochs, while when it is set to "True", it decreases with a cosine trend. The particular trend followed by the training in this work is the shown in the figure below.

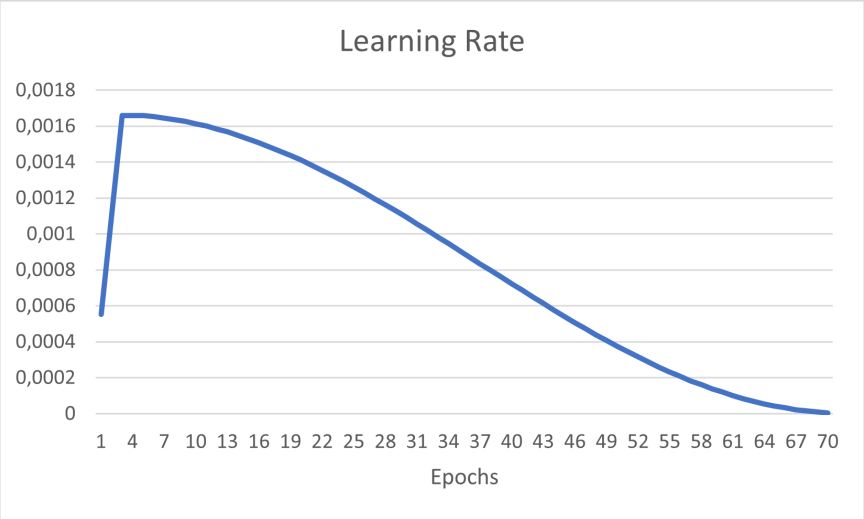


Figure 3.11: The learning rate schedule chosen for this work is in a cosine design. The increase in the first 3 epochs is caused by the warmup phase, which increase the stability of the model in the early stages.

Finally, the *close_mosaic* parameter refers to the number of final epochs, for which to disable mosaic augmentation. The latter is a technique that involves creating new training samples by combining four different images into a single mosaic image and then it is used as an input for model training. This technique helps improving the model's ability to generalize, thus helps to reduce the overfitting, especially when working with limited training data. For what regards the loss function used by YOLOv8 is the Binary Cross Entropy (BCE) loss, which measures the difference between predicted binary

outcomes and actual binary labels. It quantifies the dissimilarity between probability distributions, aiding model training by penalizing inaccurate predictions. Now it is possible to show how the training went.

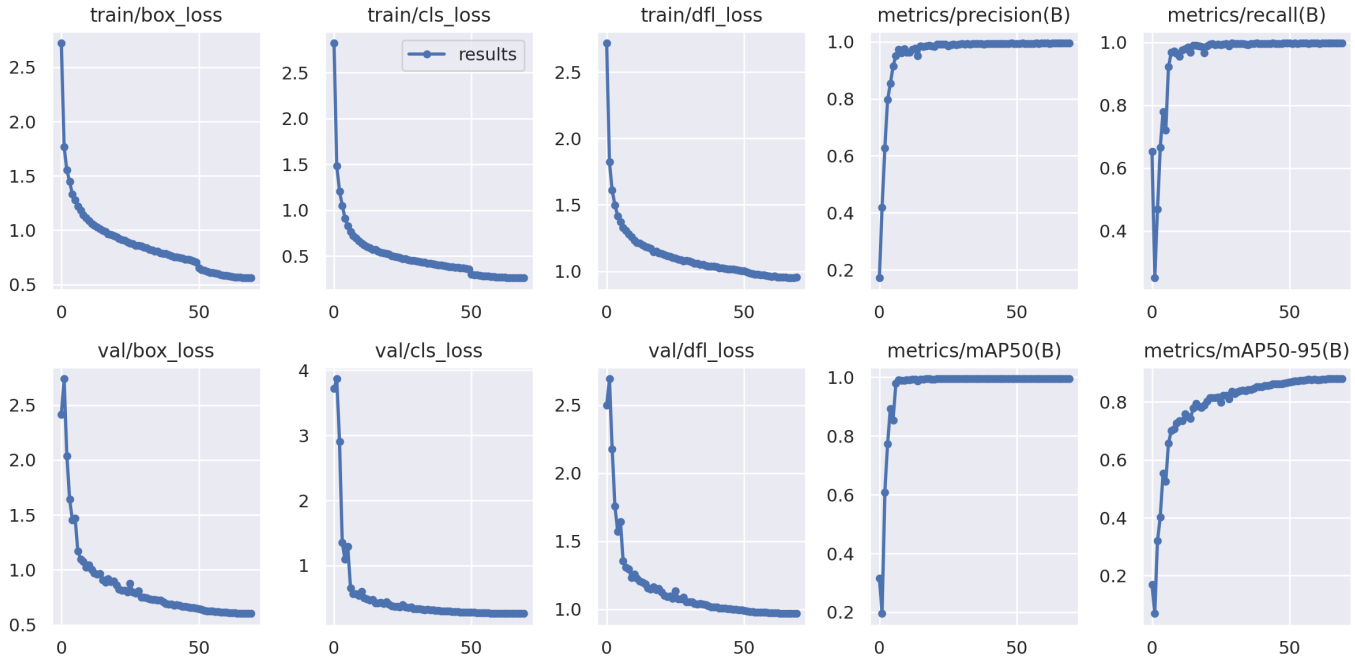


Figure 3.12: In this figure are shown the training graphs concerning the unchanged architecture of YOLOv8. They report the training and validation losses and the metrics trend over the 70 epochs.

In the previous graph, it is possible to note three different kinds of loss. In particular, there are a "box loss", a "cls loss" and a "dfl loss". The former refers to the loss calculated over the bounding boxes, the second kind is the classification loss, while the latter refers to distribution focal loss, which takes care of the imbalanced classes and aims at making sure that the model correctly detects the rare objects of these classes. The meaning of the metrics shown in the other four graphs are discussed in section 5.1.

As we can see in figure 3.12, both the training and the validation losses (in all the three categories of loss) decrease throughout the training, meaning that the network is learning deeply the features of the input data. Moreover, it can be noted the absence of overfitting, which shows when, after a certain epoch, the validation loss starts to increase instead of constantly decreasing: this means that the network is still able to generalize well on new data. For what concerns the metrics, it can be noted that, over the course of the

training, they increase, denoting an improvement in the network's ability to detect. In chapter 5, will be exposed the results achieved by this network on the test set, meaning on completely new data.

3.3 Custom

Included among the purposes of this thesis was that the object detection algorithm that would work with the software developed by VIPER [2] had to be fast enough to work in real-time applications. The network chosen, YOLOv8, is, as previously said, the current state of the art of the object detection models, both for accuracy and speed, meaning it is one of the fastest network currently available. Despite this, in order to further improve its performances in terms of speed, it was decided to modify the original architecture of YOLOv8, so that to make it lighter. In this way the inference time should reduce. As said before, the Ultralytics group - developer of YOLOv8 - have made sure that the architecture is easily scalable, thus making changes to the architecture is quite easy. Therefore, starting from the predefined factors shown in table 3.2, different attempts were made in modifying them, such as modifying just one of them per time, modifying them both together, and obviously different values have been tried. At the end, the most satisfactory combination between the depth and the width of the network, based on a compromise between performances (which obviously decreased, having the network fewer parameters) and speed (prioritized in this choice) is the following:

$$\textbf{depth} = \textbf{0.01} \text{ and } \textbf{width} = \textbf{0.01}$$

which led to a reduction in the network parameters, going from about 3 million parameters to 158 thousand and a number of layers that went from 225 to 211, as we can see in table 3.4.

Table 3.4: Comparison between YOLOv8 and YOLOv8 modified architectures.

	Standard	**Modified**
Depth	0.33	0.01
Width	0.25	0.01
Parameters	3011238	158670
Layers	225	211

For what concerns the dataset used with the modified architecture, it was decided to use the exact same dataset that was employed with the original architecture, so that a comparison could be made.

At this point of the discussion, the training hyperparameters that were fitted for this new architecture can be shown. They are summarized in table 3.5.

Table 3.5: Hyperparameters chosen for the training of the modified YOLOv8 network.

Hyperparameter	**Value**	**Purpose**
epochs	100	Number of epochs to train for.
imgsz	800	Size of input images.
batch	64	Number of images per batch.
lr0	0.001667	Initial learning rate.
lrf	0.001	Final learning rate ($lr0 * lrf$).
warmup_epochs	3	Number of epochs in which the learning rate increase, before starting decreasing.
optimizer	AdamW	Kind of optimizer to use during training.
cos_lr	True	Whether to use a cosine schedule for the learning rate during train.
close_mosaic	100	Number of final epochs, for which to disable mosaic augmentation.

In this case, the training was carried on for 100 epochs, since the network was still able to generalize, showing no signs of overfitting. The other parameters that maximized the performance of the modified architecture were the same as the unchanged architecture and so they were kept the same.

Now it is possible to show how the training went, in figure 3.13.

As previously said, the network doesn't show any sign of overfitting during the training phase, since the validation losses decrease over the 100 epochs. For what concerns the metrics, they increase but slowly when compared to the unchanged architecture, this is due to the fact that the modified architecture is much smaller than the other one and so it takes more time to learn the same features. As for the unchanged architecture, in chapter 5 will be exposed the results achieved on the test set.

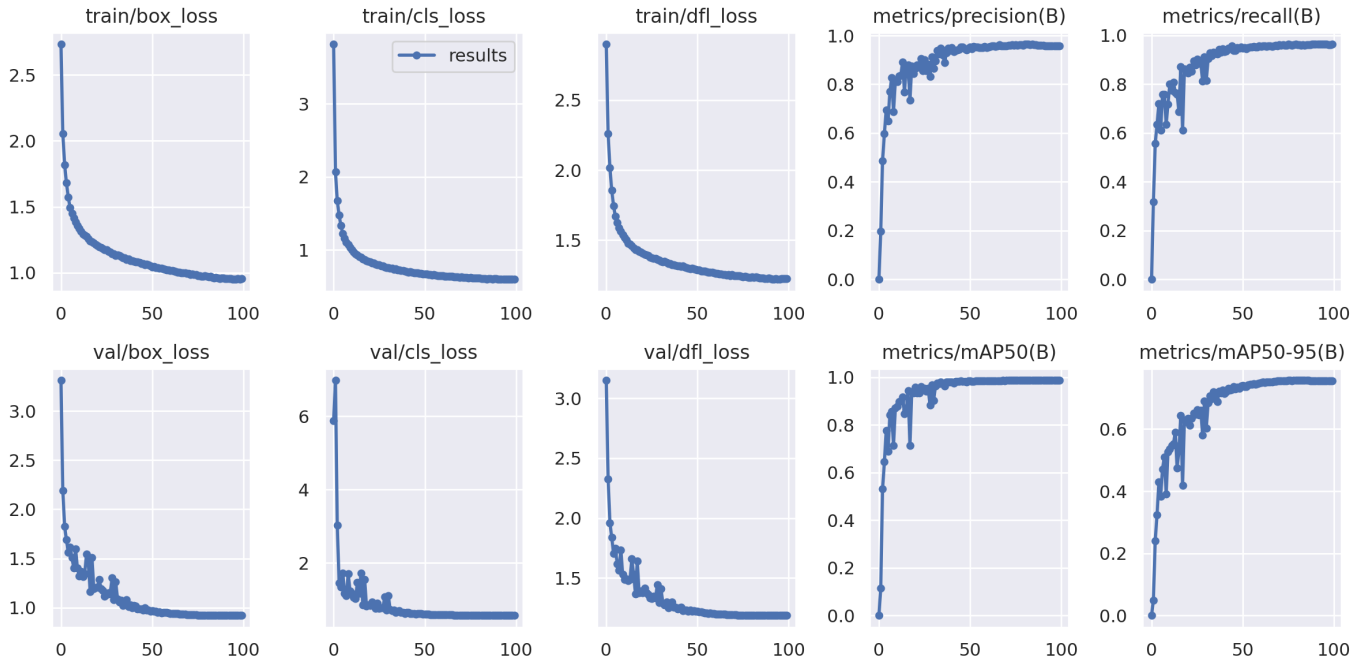


Figure 3.13: In this figure are shown the training graphs concerning the modified architecture of YOLOv8. They report the training and validation losses and the metrics trend over the 100 epochs.

Chapter 4

DeepSORT

4.1 Object Tracking: an overview

Object tracking is a computer vision technique that is usually coupled with object detection algorithms, with the goal of augmenting their capabilities: while object detection focuses on identifying and localizing objects within images or video frames, object tracking extends this functionality by assigning unique IDs to the detected objects and following them across multiple frames, allowing the monitoring and analysis of object movements. It is clear that the main purpose of object tracking is to maintain a continuous association between detected objects in consecutive frames, in such a way that it can handle hard situations like occlusions, scale variations, object deformation and changes in appearance due to different lighting conditions, in which object detection could suffer.

As said before, object tracking is very useful when the object finds itself in situations where it can be difficult to detect, for this reason object tracking finds many applications in fields such as autonomous vehicles, surveillance, robotics and augmented reality. There are two main types of object tracking: single-object tracking and multi-object tracking. The first one focuses on detect just one object throughout the sequence of frames, resulting in a very fast method. Some examples of single-object tracking algorithms are CSRT and KCF, which are traditional computer vision based; or SiamRPN and GOTURN, which instead are deep learning based. Multi-object tracking methods can detect more than one object simultaneously in the same video-frame and belonging to different classes: this method is more accurate compared to single object tracker. Some examples of this method are DeepSORT, JDE and CenterTrack.

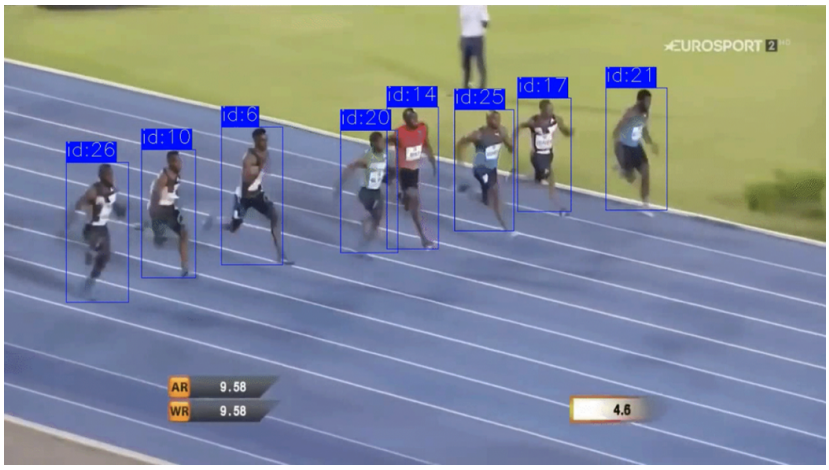


Figure 4.1: Object tracking assigning unique IDs to different objects detected.

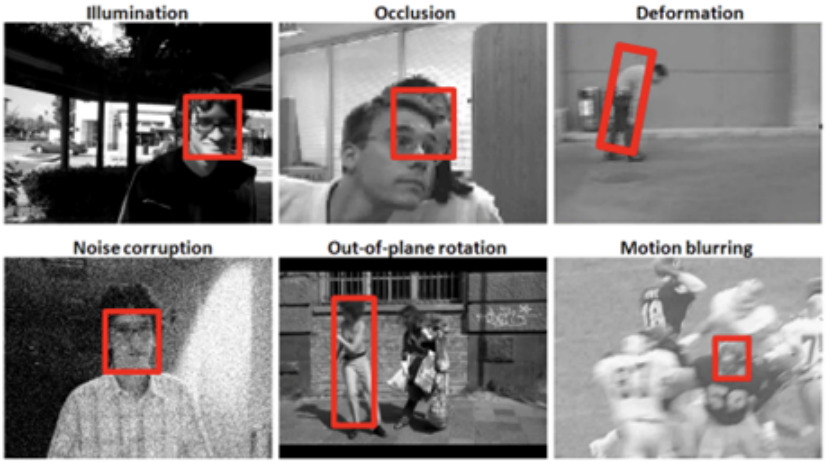


Figure 4.2: Examples of how object tracking algorithm should deal with challenging situations, like occlusions, deformations, illumination changing, noise corruption, out-of-plane rotation, motion blurring [22].

In figure 4.3 it is represented a general flow of how an object tracking algorithm works during the inference process.

First, an input video file or a real-time feed from a camera is fed into an object detection algorithm: the latter will perform detections on the input, based on what it learnt during the training phase. The object detection algorithm will output a series of bounding boxes around the objects it detected and classified in each frame. In the last step, the bounding boxes outing from the object detection model are fed into the object tracking algorithm, which

Figure 4.3: Flowchart of the object tracking coupled with object detection.

will keep track of the detected object through consecutive frames.

Despite the need to track one single object, which is Inferior Vena Cava, in this work of thesis it has been chosen to use DeepSORT as object tracking algorithm, for a series of reasons strictly bonded to the intrinsic nature of ultrasound videos, for example the latter can be characterized by suboptimal image quality or the IVC may be partially occluded by other anatomical structures throughout the video or it can change in scale as the ultrasound probe moves, so having a tracking algorithm that offers a high level of tracking accuracy and the ability to handle challenging scenarios is crucial in this context, and DeepSORT represents the state-of-the-art of the object tracking algorithms. Furthermore, it is a very fast algorithm, such that can work fine in real-time applications: the latter is a fundamental characteristic to search for in detection and tracking algorithms because they have to work in real-time with the software developed by VIPER s.r.l. [2] in order to obtain measurements like diameters, areas, caval indexes and so on.

4.2 Architecture

DeepSORT (deep Simple Online and Realtime Tracking) [23] is one of the most popular object tracking algorithms used nowadays. It was introduced by Wojke et all. in 2017 and it is an extension to SORT (Simple Online and Realtime Tracking) [24], which is a simple framework, based on the Kalman filter that performs object tracking, using traditional methods. The main limitation of SORT is the ease with which identity switches occur. So, in order to solve this issue, DeepSORT integrates an appearance descriptor, allowing it to track objects for longer periods of occlusions, reducing in this way identity switches [23], hence making tracking more efficient. Furthermore, DeepSORT is deep learning based, which means that it is an algorithm that can be trained offline, during which it learns a deep association metric.

Before discussing in detail how DeepSORT operates, it is necessary to firstly introduce its main features, which are Kalman filter, the Hungarian algorithm and the appearance descriptor.

Kalman filter The Kalman filter is an algorithm that tracks the position and motion of an object in subsequent frames, basing its prediction on the object's position and motion in previous frames. Basically, it combines the information from past frames and uses them to estimate the position and motion of the object in the current frame. To do so, it is used an eight dimensional vector that describes the state space of a detected object and it is defined as $(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ where (u, v) are the bounding box center coordinates; γ is the aspect ratio, h is the height and $(\dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ are the respective velocities in image coordinates [23]: in this way the positional coordinates, meaning the bounding box describing the object, are used to predict position and velocities to predict motion in the next frame.

Hungarian algorithm Now that we have the prediction of the Kalman filter, it is necessary to assign the prediction to newly arrived measurements: to do this, Hungarian algorithm is used. In particular, it is necessary to integrate motion and appearance information. For what concerns the motion information, it is used the Mahalanobis distance between the Kalman state and the new detections in the current frame:

$$d_{(i,j)}^{(1)} = (d_j - y_i)^T S_i^{-1} (d_j - y_i) \tag{4.1}$$

Where (y_i, S_i) refers to the i-th track of the Kalman state, while d_j refers to the new detected object. The Mahalanobis distance takes state estimation uncertainty into account by measuring how many standard deviations the detection is away from the mean track location [23].

To take into account the appearance information, it is used a second metric that uses the appearance descriptor. This second metric measures the smallest cosine distance between the $i-th$ track and $j-th$ detection in appearance space:

$$d_{(i,j)}^{(2)} = min\{1 - r_j^T r_k^{(i)} | r_k^{(i)} \in \mathcal{R}_i\} \tag{4.2}$$

where r_j is the appearance descriptor and $\mathcal{R}_k = \{r_k^{(i)}\}_{k=1}^{L_k}$ is a gallery of the last $L_k = 100$ associated appearance descriptors for each track k. The appearance descriptors are obtained through a pre-trained CNN, which will be described in the next paragraph.

In conclusion, the two described metrics cooperates to solve the assignment problem: the Mahalanobis distance is valuable for estimating potential object positions based on motion, making it well-suited for short-term predictions.

In contrast, the cosine distance focuses on appearance information, which is particularly valuable in case of the need to recover identities after extended occlusions, when motion is less reliable. At the end, the two metrics are combined through a weighted sum.

In the following flowchart is shortly summarized how an object detection algorithm (in this case YOLOv8) and DeepSORT cooperate to perform object tracking.

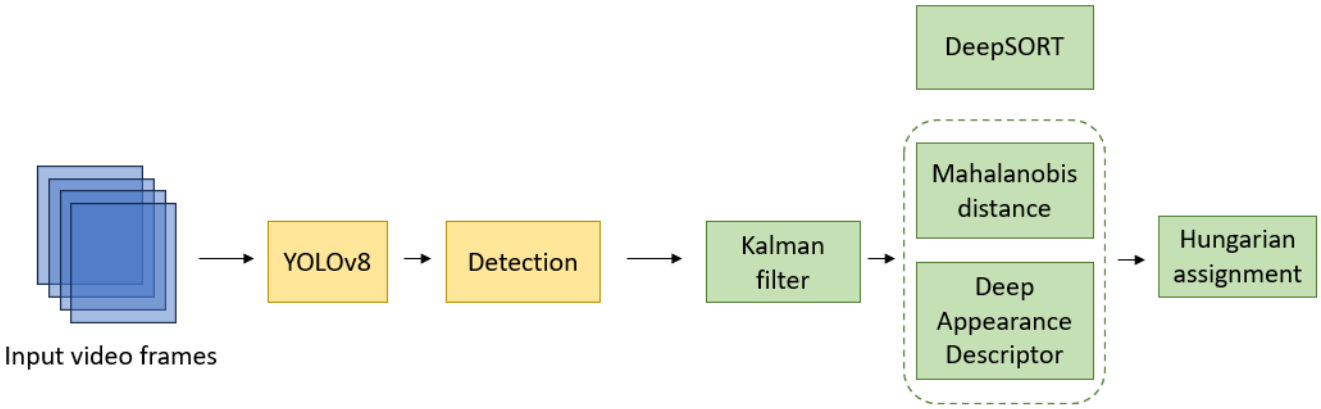


Figure 4.4: Flowchart of how YOLOv8 and DeepSORT work together.

4.3 Training

As for the YOLOv8 network, also for DeepSORT, before entering in the details of what training strategies have been adopted in this work of thesis, it will be discussed the way in which the dataset has been obtained.

Dataset

In order to train DeepSORT, it cannot be used the same dataset used to train YOLOv8, because, as the latter needed the data in a specified format, also DeepSORT needs its data in a specified format. In particular, DeepSORT doesn't need labels in text files, but it needs cropped images in which is contained only the object to detect, may it be full or just parts of it. Keeping this in mind, it has been decided not to start from scratch for generating a proper dataset for DeepSORT, but to start from the labeled YOLOv8 dataset. In particular, it has been written a Python code that, starting from the full image, crops the latter in correspondence of the bounding box, so that

to obtain an image with the specific size of 64x128 required by DeepSORT and in which the object to be detected is present for sure. An example of the obtained images is reported in the following figure:

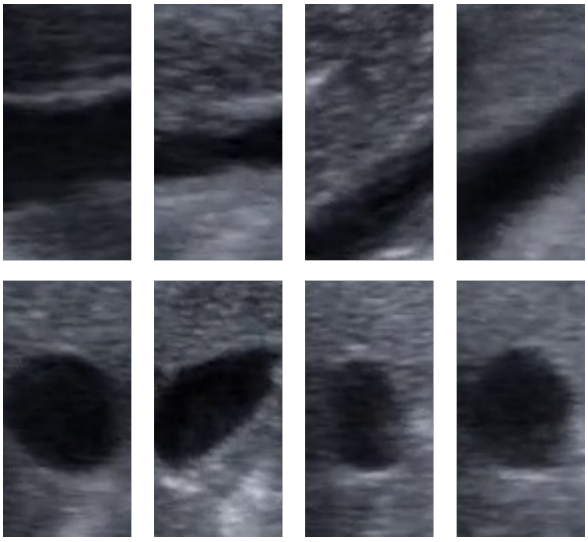


Figure 4.5: Examples showing samples from the DeepSORT dataset. In particular on the top line are shown examples of the longitudinal IVC, while on the bottom line are reported examples of transversal IVC. In both cases the dimension must be $64x128$.

For what concerns the knowledge of the classes of the different images, the latter have to be stored in different folders, named as the class you want to detect. In the next paragraph, it will be discussed the training of DeepSORT.

Training

The training of DeepSORT has been done using the same hardware used with the training of YOLOv8, which is summarized in table 3.1. Regarding the final dataset used, it was composed of a total of 8891 images for the training set and 8803 images for the test set.

While, for what concerns the hyperparameters set for the training, also in this case it has been done a finetuning work to discover the best combination of hyperparameter that guarantee the best performances possible and, at the end, it has been chosen to use the default combination, because, as said before, it has shown to bring the best performances. The combination chosen is then reported in the table below. At his point of the discussion, it is possible to show how the training of DeepSORT went: it is shown in figure 4.6.

Table 4.1: Hyperparameters chosen for the DeepSORT training.

Hyperparameter	**Value**	**Purpose**
epochs	50	Number of epochs to train for.
lr	0.1	Starting learning rate.
optimizer	SGD	Optimizer used: Stochastic Gradient Descent.
max_dist	0.2	Maximum Mahalanobis distance for associating detections to existing tracks.
min_confidence	0.3	Minimum confidence score required for a detection to be considered valid.
NMS_max_overlap	0.5	Maximum allowed overlap between bounding boxes during Non-Maximum-Suppression.
max_iou_distance	0.7	Maximum intersection-over-union (IoU) distance between two bounding boxes for them to be considered as the same object.
max_age	70	Maximum age of a track (in frames) before it is considered for deletion.
n_init	3	This is the number of consecutive detections that are required to establish a new track.
nn_budget	100	Maximum number of candidates that are considered when associating a detection with an existing track.

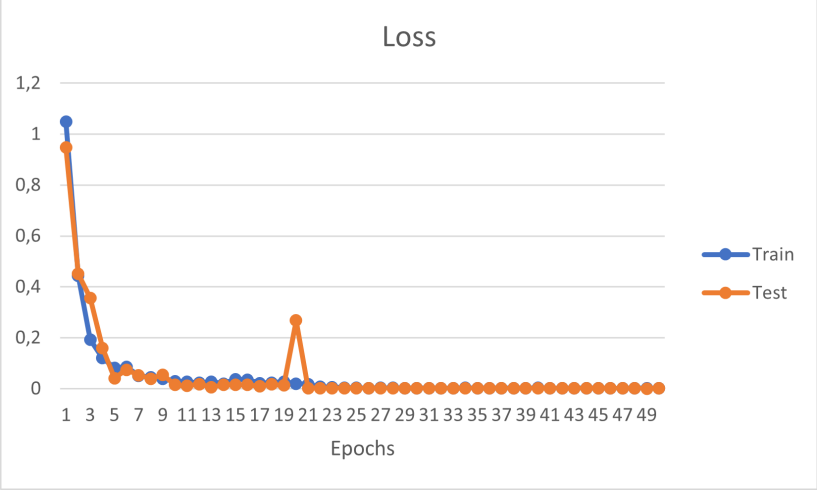


Figure 4.6: Training and test losses reported during the DeepSORT training.

As we can see in the diagram, both the training and test losses decrease during the training phase, as we expect, suggesting an ever-improving network. Moreover we can see that the two trends showed a plateau, meaning that the network has learnt all it could learn from the data. The performances of this network coupled with the two YOLOv8 architectures are shown in the chapter 5.

Chapter 5

Results

After exposing the methods used in this work of thesis, in terms of choice of the networks and training strategies applied, in this last chapter they will exposed the results that were achieved. However, before showing that, it will be exposed a brief overview of the metrics used to evaluate the performances of the trained networks. In particular, they will be discussed both the object detection and the object tracking metrics so that we can distinguish between the performance of the two YOLOv8 architectures and the complete algorithm with DeepSORT separately.

5.1 Metrics

5.1.1 Object detection metrics

Mean Average Precision (mAP) is a widely used metric for evaluating the robustness of object detection algorithms. It calculates the Average Precision (AP) for all-classes in a multi-class problem. AP is defined as follows:

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k) \tag{5.1}$$

In the equation 5.1, $P(k)$ is the Precision in the data point k, while $\Delta r(k)$ is the Recall in the same point. It is therefore understood that to fully understand mAP, it is necessary to understand what Precision and Recall are. They are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

In the previous equations 5.2 and 5.3, TP are True Positive, FP are False Positive and FN are False Negative. These last terms are defined in the classification field and are better explained through a confusion matrix:

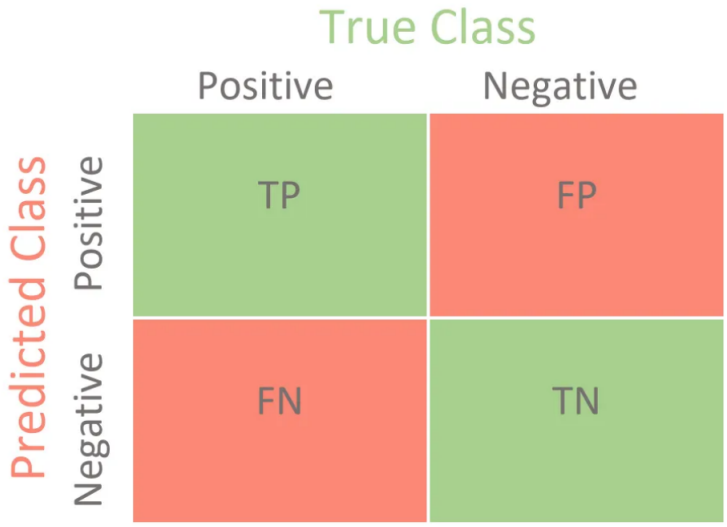


Figure 5.1: Representation of a general confusion matrix: on the diagonal are reported the correct outcomes of the network, while in the other spots are reported the mistakes.

Therefore, considering a classification task, True Positives (TP) and True Negatives (TN) are the samples that were correctly classified, as we can see in green in the diagonal in figure 5.1, while False Negatives (FN) are the samples classified as negative, but that were actually positive; on the other hand, False Positives (FP) are the samples classified as positive, but that were actually negative. Keeping in mind this distinction and looking at the expression of Precision and Recall, we can state that Precision is defined as the ratio between those correctly classified as positive (TP) and the totality of those classified as positive and so it answers the question: "How many of those classified as positive were really positive?". Whereas, Recall is defined as the ratio between those correctly classified as positive (TP) and the totality of authentic positives, and so it answers the question: "How many objects were correctly classified as positive?". However, unlike the classification task, in which an item is undoubtedly classified as positive or negative, in the detection task, the distinction between positive and negative is not

so clear-cut. To determine whether an object is correctly detected or not, a threshold needs to be set. This threshold is defined by the Intersection over Union metric. The Intersection over Union (IOU) is a widely used metric for evaluating the overlap between two bounding boxes: the predicted bounding box and the ground truth. A visual representation of IOU is shown in figure 5.2.

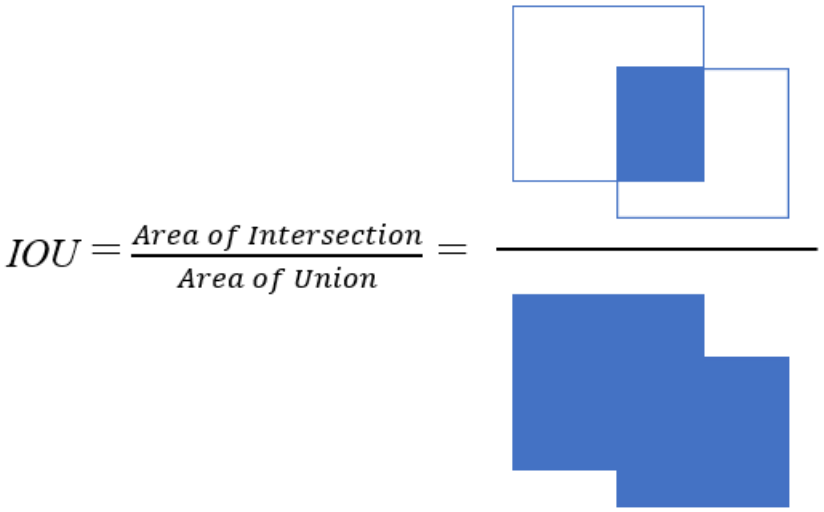


Figure 5.2: Visual representation of IoU metrics.

As we can see in figure 5.2, by evaluating the ratio between the intersection and the union between the predicted bounding box and the ground truth, IOU is a measure of the accuracy of the prediction: the closer it is to 1 more accurate the model - and its prediction - is. It means that choosing a higher IOU as threshold will lead to a stricter distinction, while a lower IOU means a milder distinction between correct and incorrect detections. In this thesis, to evaluate performances, it has been chosen an IOU of 0.5.

For the evaluation of YOLOv8 performances, it has been used a particular kind of mAP, which is mAP50-95. The latter measures the mAP of examples across a range of IOU thresholds, ranging from 0.5 to 0.95. In this way, a more comprehensive evaluation of performances is provided, since it considers different levels of overlap between the two bounding boxes.

5.1.2 Object tracking metrics

The metrics used to evaluate object tracking algorithm's performances are supplied by MOTmetrics library [25]. The latter provides a wide range of metrics and, for the purpose of this thesis, it was decided to use the following metrics: Multi-Object Tracking Accuracy (MOTA), Multi-Object Tracking Precision (MOTP), Identification F1-score (IDF1). Before starting the detailed description of the metrics, it is worth noting that the MOTmetrics library uses a slight different version of IOU:

$$IOU = 1 - \frac{Intersection}{Union} \tag{5.4}$$

This expression has the reverse meaning of the previously described: an IOU equal to 0 means perfect overlap, while an IOU close to 1 means poor overlap.

MOTA Multiple-Object Tracking Accuracy (MOTA) is perhaps the most widely used metric to evaluate a tracker's performance and it is defined as follows:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t} \tag{5.5}$$

where t is the frame index, FN are the false negative predictions, FP are the false positive ones, $IDSW$ are the identity switches and GT is the number of ground truth objects [26]. MOTA can range from $-\infty$ to 1: MOTA values near to one means high accuracy. It is worth noting that the MOTA metric could be also negative, in cases where the number of errors made by the tracker exceeds the number of the ground truth in the scene [26]. This metric is particularly powerful due to the fact that it takes into account not one, but three sources of errors (FN, FP, $IDSW$), however it is not enough to fully comprehend the algorithm's performances: for this reason it is often coupled with other metrics, such as MOTP.

MOTP Multi-Object Tracking Precision (MOTP) is a metric the evaluate the goodness of localization precision and it is defined as follows:

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \tag{5.6}$$

where c_t denotes the number of matches in frame t and $d_{t,i}$ is the bounding box overlap of target i with its assigned ground truth object [26]. Basically it measures the average distance between the predicted and ground truth bounding boxes. MOTP ranges from 0 to 1 and, differently from the MOTA metric, a good MOTP shows values near to 0.

IDF1 The previously described metrics focus on a detection aspect of the tracking, evaluating the tracking accuracy and the localization precision. Beyond this aspect, it is also important to evaluate the identification capability of the algorithm. To do so, MOTmetrics library [25] supplies a series of different metrics. The one used in this thesis is the Identification F1-score (IDF1), which measures the tracker's ability to maintain correct target identities over time. IDF1 score is described by the following expression:

$$IDF1 = \frac{2IDTP}{2IDTP + IDFP + IDFN} \tag{5.7}$$

in which IDTP (Identity True Positive) corresponds to the longest associated trajectory matching to a ground truth trajectory, while every other trajectory matching the same ground truth trajectory is targeted as IDFP (Identity False Positive); IDFN (Identity False Negative) are all the ground truth trajectories that remain unmatched [27].

The IDF1 metric combines identity-based precision (IDP) and identity-based recall (IDR): these metrics have the same meaning of the ones described in section 5.1, but applied to the identities. Their expression are newly reported as follows:

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{5.8}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{5.9}$$

5.2 Testing

In this section, they will be exposed the results achieved in this work of thesis, through a testing of the networks trained, both the YOLOv8 models, the one with the original architecture (which we will refer to as *original*) and the one with modified architecture (which we will refer to as *modified*) and the DeepSORT. In the following it will be made a distinction between the

performances of the object detection models and the object tracking model coupled with the two different YOLOv8 models. Finally it will be tested the possibility to integrate the algorithms with the software developed by VIPER [2].

Object Detection Performance

For what regards the performances of the detection models, they are exposed in table 5.1, in which are compared the mAP50-95, whose meaning is discussed in the previous section, of the two architectures considered. In order to have a clearer vision of the performance, in the table are reported the performances on the two different classes (IVC long and IVC trasv, which refer to longitudinal IVC and transversal IVC respectively) and the overall metrics.

Table 5.1: Object detection performances of both the original YOLOv8 model and the modified one.

	mAP50-95 (%)	
	Original	**Modified**
IVC long	91.6	77.2
IVC trasv	84.8	69.8
Overall	88.2	73.5

As we expected the original YOLOv8 model showed better performances compared to the modified one, in fact, while the former reaches 88.2% of overall mAP, the modified one only reaches 73.5%. This is mainly due the fact the modified architecture is significantly smaller than the original one. However, for this same reason the modified architecture showed an inference time which is, as we expected, lower than that showed by the original one, as we can see in the following table:

Table 5.2: Mean inference time of the two architectures considered.

	Original	**Modified**
Inference time	12.1ms	10.4ms

In the following, will be showed some examples of detections performed by the original YOLOv8 and the modified YOLOv8, in order to make not

only a quantitative analysis but also a qualitative analysis of the abilities of the two networks.

In figure 5.3, it is reported a comparison of detections performed by the two architectures: in the left column are reported the detections operated by the original architecture, while on the right there are the detections performed by the modified one. Obviously, in order to make a meaningful comparison, the same image has been used for the two architectures. As can be noted, the original architecture performs a much better detection compared to the modified one, as the table 5.1 suggests. The original one shows, for every image, a confidence score higher than 80% (sometimes reaches even 90%), while the modified one shows different kinds of errors: ranging from not identifying the IVC at all, to find two of them in the same frame. Nonetheless, it can also perform good detections, even with a lower confidence score, like suggests the third image.

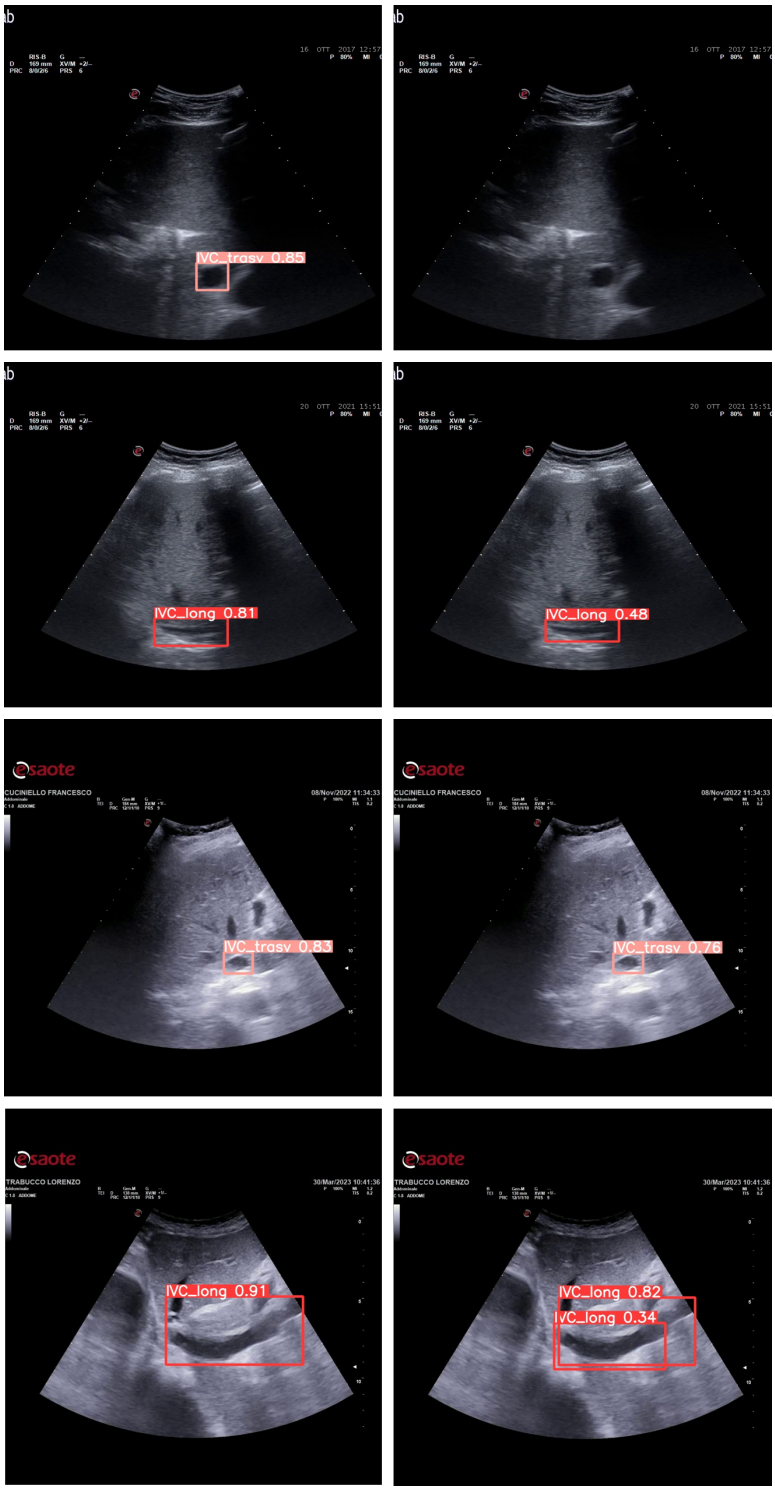


Figure 5.3: Examples of detections performed on several images by the two architectures: in the left column are reported the detections operated by the original architecture, while on the right there are the detection performed by the modified one. In addition, it is reported also the confidence score of the detections.

Object Tracking Performance

In this section, are reported the object tracking performances, meaning the performances reached by the two architectures when coupled with an object tracking algorithm, which is, in this case, DeepSORT. The achieved results were calculated on a set of videos, which the two networks have never seen. The results are reported in tables 5.3 and 5.4.

Table 5.3: Object tracking performances of the original YOLOv8 model.

	False positives	**Precision (%)**	**Recall (%)**	**MOTA (%)**	**MOTP (%)**	**IDP (%)**	**IDR (%)**	**IDF1 (%)**
IVC long	27	89.8	81.0	72.6	24.5	87.3	79.7	82.6
IVC trasv	10	93.3	75.1	71.3	19.8	92.1	74.2	80.4
Overall	20	91.2	78.5	72.0	22.6	89.3	77.4	81.7

Table 5.4: Object tracking performances of the modified YOLOv8 model.

	False positives	**Precision (%)**	**Recall (%)**	**MOTA (%)**	**MOTP (%)**	**IDP (%)**	**IDR (%)**	**IDF1 (%)**
IVC long	122	76.3	76.2	50.5	29.0	63.5	63.4	63.4
IVC trasv	116	65.7	56.6	24.5	20.1	55.5	46.8	50.4
Overall	119	71.9	68.0	39.7	25.3	60.1	56.5	58.0

As we can see in the tables, the original architecture of YOLOv8 has reached higher performances in the tracking case too. Moreover, as we can see from the first metric reported, the original architecture presents a much lower number of false positives compared to the modified architecture. False positives are defined when the network finds an object where it is no object to detect, for example in figure 5.3, in the last image, the modified network finds two IVCs, when there is obviously just one. This makes the original architecture more reliable: it presents also a much higher MOTA and a higher MOTP (which, as described in the previous section, the lower it is the better is the precision), even if, in this case, the difference is not so evident, since the original one has a 22,6%, while the modified one shows 25,3%.

For what regards the IDF-1 metric, the difference between the two network is very high, in fact if the modified network shows a 58%, the original one exceeds 81%: this means that the latter has a better ability to maintain correct target identities over time.

Coupling with VIPER software

In order to verify that the trained networks can work properly with the software developed by VIPER, it has been decide to add another metric for evaluating the performances. It has been decided to assess the percentage of frames in which the center of the bounding box coming from the algorithm, falls into the segmentation performed by the VIPER software.

In the following table, are reported the results.

Table 5.5: Performance to evaluate the possibility of coupling the proposed algorithm with the VIPER software.

	Original (%)	**Modified (%)**
IVC long	76.77	55.03
IVC trasv	98.88	79.61
Overall	87.82	67.32

The previous table suggests excellent possibilities of coupling the proposed algorithm with the software developed by VIPER, especially in the case of the original architecture, which has shown very high performances (almost approaching the 90% on the overall case) in both the two classes implied in this work. The performances characterizing the *IVC long* class are lower than the *IVC trasv* class, because the segmentation operated by the VIPER software doesn't involve the integrity of the IVC but just a section of it, impling that in certain frames the center of the bounding box may slide out of the segmentation, but always remaining inside the inferior vena cava, as is shown in the second image of figure 5.4.

In the following figure, are shown some examples of the coupling of the two algorithms with the VIPER software, in particular, it is reported the segmentation - in red - and the center of the bounding box - in yellow -, to verify that it falls indeed inside the segmentation.

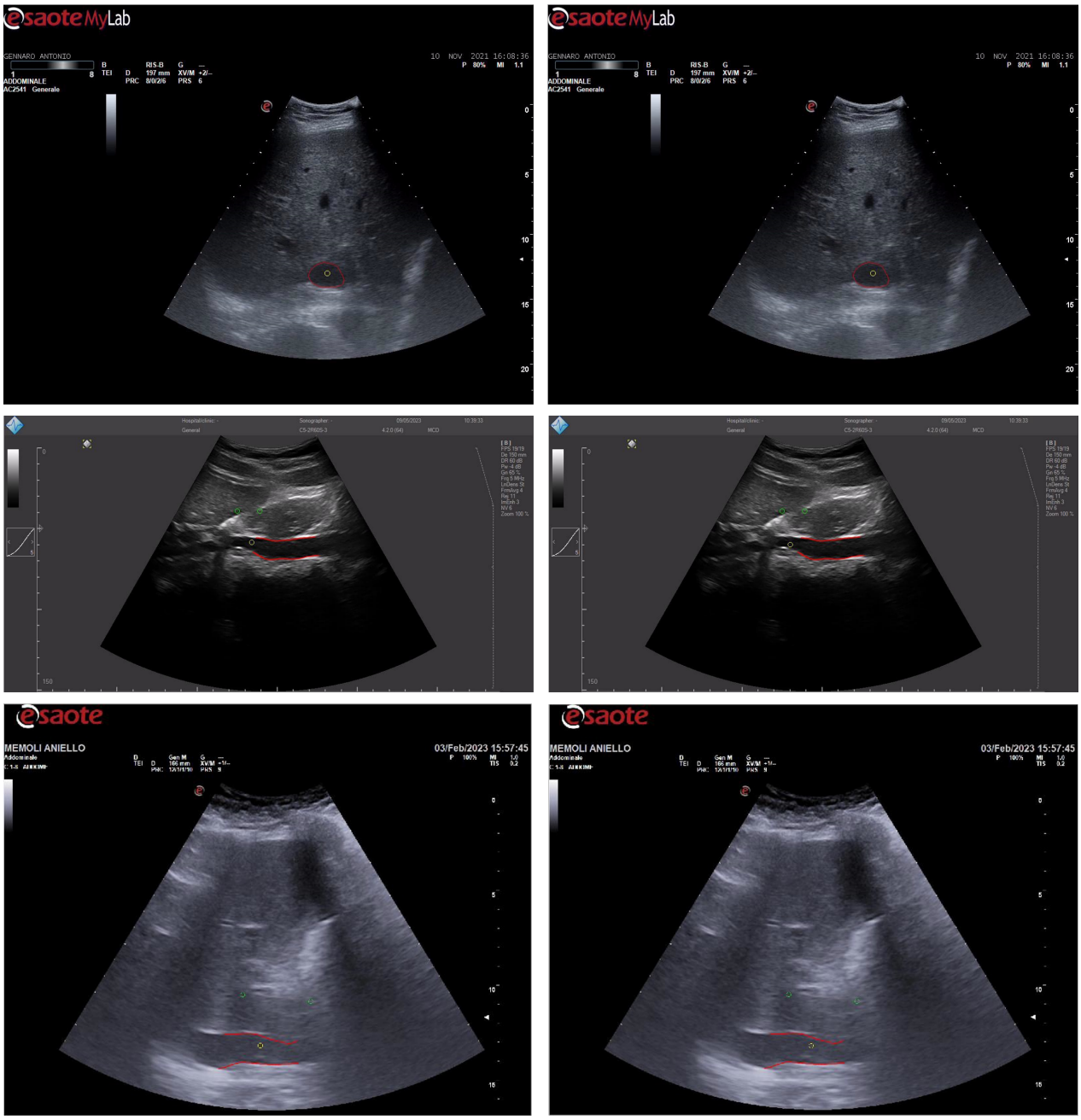


Figure 5.4: Examples of the comparison between the outcome of the detection algorithms and the segmentation by the software developed by VIPER. In the left column are reported the frames relative to the original architecture, while on the right column the ones relative to the modified architecture. In red is reported the segmentation, while in yellow is reported the center of the bounding box.

5.3 Eigen-CAM

After exposing the achieved results in this work of thesis, it is interesting to look at the predictions under another aspect: what brought the object detection network to individuate the objects in the image. To do so, explainability methods can be employed. The latter, in fact, report, through a color map, what pixels in particular were responsible for the prediction made, giving us a very intuitive way to better understand how a deep learning algorithm works. There are different techniques that have the same purpose and they all are based on the concept of Class Activation Map (CAM) - firstly introduced in 2015 [29] - which is a method specifically designed for convolutional neural networks. This method computes the class activation map, based on three processes: removal of the fully connected layers at the end of the CNN; replacement of the MaxPooling layer by a global average pooling layer; computation of the weighted average of features extracted by the feature extraction network [30]. Examples of newer techniques based on CAM are Grad-CAM, Grad-CAM++, CNN-fixations and Eigen-CAM. In particular, in this work of thesis it was employed the latter.

Eigen-CAM is a method introduced in 2020 [30] and it eliminates the need to modify CNN architecture or to backpropagate some computations. In fact, it assumes that all relevant spatial features in the input image learned over the hierarchy of the CNN model will be preserved during the optimization process, and non-relevant features will be regularized or smoothed out [30]. How the Eigen-CAM is obtained is shown in the following. Let's consider an image I with dimensions i, j and the combined weight matrix $W_{L=n}$ of the first k layers of size (m, n), then [30]:

$$O_{L=k} = W_{L=n}^{T} I \tag{5.10}$$

is the class activated output, obtained by projecting I onto the last convolutional layer $L = k$. Now $O_{L=k}$ can be factorized using singular value decomposition, in order to obtain its principal components:

$$O_{L=k} = U \Sigma V^{T} \tag{5.11}$$

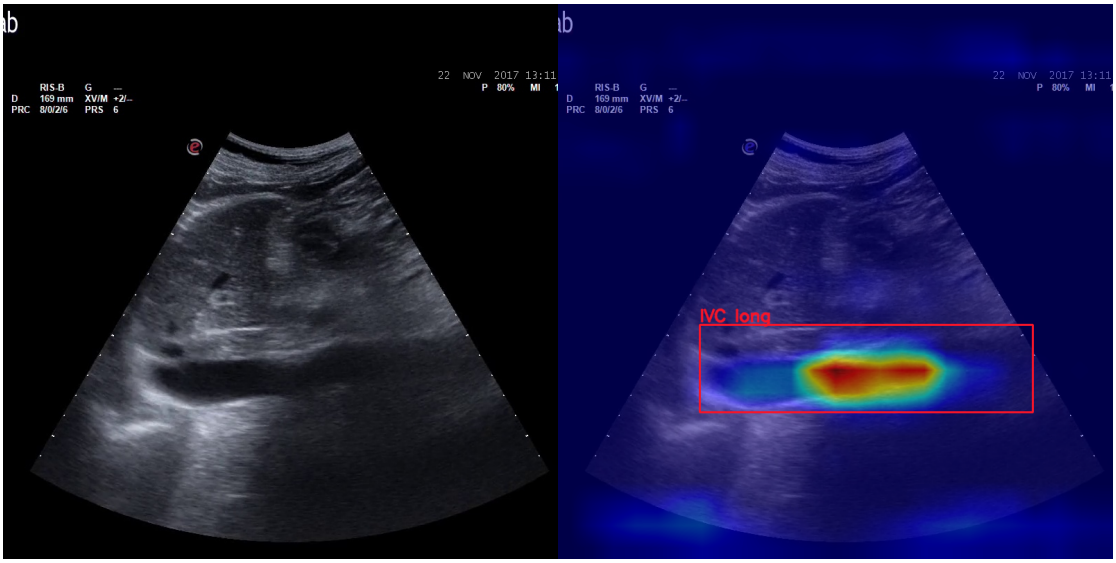
where U in an MxM orthogonal matrix and its columns are the left singular vectors, Σ is an MxN diagonal matrix, in whose diagonal there are singular values, while V is an NxN orthogonal matrix with left singular vectors on its columns. Thereby, the class activation map, meaning the Eigen-CAM is given by the projection of $O_{L=k}$ on the first eigenvector [30]:

$$L_{Eigen-CAM} = O_{L=k}V_1 \tag{5.12}$$

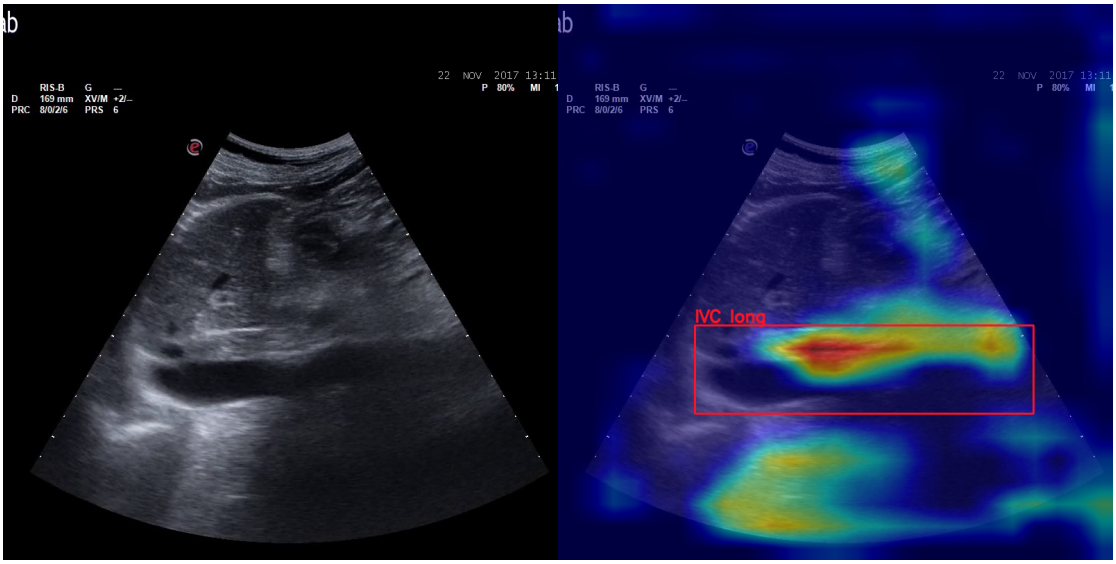
In figures 5.5 and 5.6 it is shown an example of Eigen-CAM applied to this thesis. In particular it is represented an inferior vena cava both in longitudinal (figure 5.5) and transversal (figure 5.6) section: on the left of both pictures is reported the original image, while on the right side is present the elaborated images, in which are present both the prediction of the network (bounding boxes denominated respectively *IVC long* and *IVC trasv*), and the Eigen-CAM map. As previously anticipated, the latter is represented through a color map, that employs a color scale, ranging from blue to red, in which blue means "low value" and red "high value". In particular, in the case of object detection, the blue zones indicate pixels that have contributed little in the prediction, whilst red zones indicate pixels that have deeply contributed in the prediction.

Let's consider the top images of figures 5.5 and 5.6, which means the ones relative to the original architecture. It can be noted that the red zones falls pretty well inside the IVC, meaning that the network is able to individuate and classify correctly the inferior vena cava, both in longitudinal and in transversal section, relying only the IVC features and not, for example, on its surroundings. If now, we compare these predictions with the ones outing from the modified architecture, it can be noted that they are not so sharp as the ones made by the original architecture, in fact the network relies, not only on the IVC (in particular the network highlights the edges of the IVC on both cases), but also on its surroundings: it is evident in the case of the longitudinal IVC.

In conclusion, it can be said that the study of the Eigen-CAM applied on the two networks of this thesis suggests that the original architecture supply a more reliable prediction than the one supplied by the modified network.

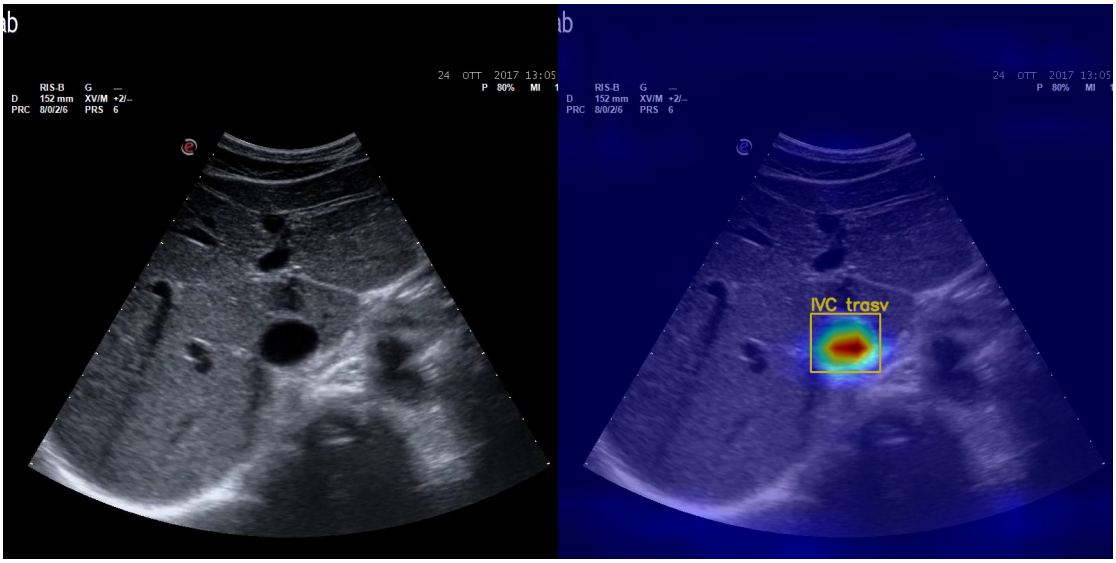


(a) *Original architecture*

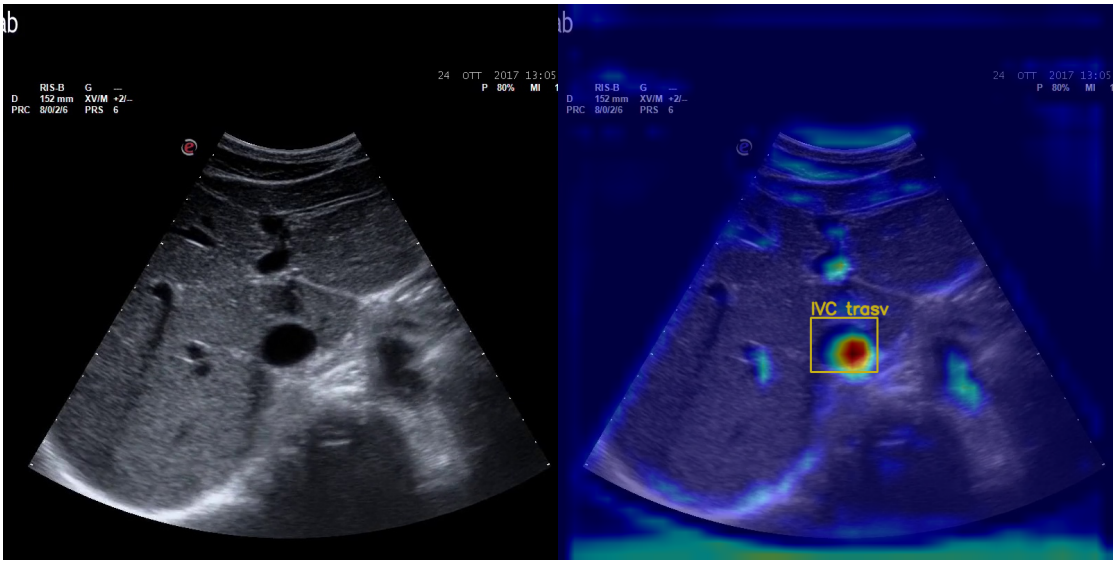


(b) *Modified architecture*

Figure 5.5: Example of the Eigen-CAM applied to one image representing the longitudinal IVC. In the top image is reported the Eigen-CAM relating to original architecture, while in the bottom image is reported the Eigen-CAM relating to the modified architecture.



(a) *Original architecture*



(b) *Modified architecture*

Figure 5.6: Example of the Eigen-CAM applied to one image representing the transversal IVC. In the top image is reported the Eigen-CAM relating to original architecture, while in the bottom image is reported the Eigen-CAM relating to the modified architecture.

Chapter 6

Conclusions

In this work of thesis, it has been discussed the proposal of an algorithm to be coupled with the software developed by a start-up, VIPER s.r.l, that can allow it to work in real-time and will make the software operator-independent. The aforementioned software is in fact capable of performing segmentation of the inferior vena cava (IVC), with the goal of obtaining some important indexes (such as Caval Index, diameters, areas, and so on), useful for the assessment of the volemic state and the Right Atrial Pressure (RAP) of a patient, information that have implications in various fields of medicine. However, the software was still operator-dependent and so, with this work of thesis, it has been proposed an object detection algorithm to make it fully operator-independent and that can work in real-time. To achieve this goal it has been proposed the current state of the art model for object detection, which is YOLOv8, and, in order to further improve its capacity, it has been coupled with an object tracking algortithm, in particular it has been chosen DeepSORT. Furthermore, to improve the speed of detection of the network, it has been proposed a modified version of the YOLOv8 architecture, so that to make it lighter. The results achieved in this work suggest that the algorithm with the original YOLOv8 has better overall performances both in the case of object detection and object tracking, reaching a mAP of 88,2% in detection ability, a tracking accuracy of 72% and an IDF1-score of 81,7%, against 73,5%, 39,7% and 58% respectively, shown by the modified architecture. However, the latter showed an inference time shorter of 2ms compared to the original architecture. Moreover, the original architecture has shown a better coupling with the aforementioned software developed by VIPER, in fact the center of the bounding box falls into the segmentation the 87,8% of

the frames, against the 67,3% shown by the modified architecture. In conclusion, it can be said that the algorithm with the original architecture of YOLOv8 would represent a better choice to couple with the VIPER software, because the reduction of the inference time of the modified architecture does not worth the lowering of the performances reported.

Bibliography

[1] L. Mesin, "Nuove tecniche ultrasonografiche di studio della vena cava inferiore per guidare il management del paziente in diversi contesti clinici", webinar organizzato in data 9 gennaio 2021,URL:https://www.lingomed.it/fad2/wp-content/video/nuove-tecniche-ultrasonografiche/live.mp4

[2] VIPER, URL: https://www.viper.srl/

[3] S. Albani, L. Mesin, S. Roatta, A. De Luca, A. Giannoni, D. Stolfo, L. Biava, C. Bonino, L. Contu, E. Pelloni, E. Attena, V. Russo, F. Antonini-Canterin, N. Riccardo Pugliese, G. Gallone, G. Maria De Ferrari, G. Sinagra, P. Scacciatella, "Inferior Vena Cava Edge Tracking Echocardiography: A Promising Tool with Applications in Multiple Clinical Settings", diagnostics, 2022

[4] Humanitas,URL:https://www.humanitas.it/enciclopedia/anatomia/apparato-cardiocircolatorio/vene-e-sistema-venoso/vena-cava-inferiore/

[5] L. Mesin, S. Albani, P. Policastro, P. Pasquero, M. Porta, C. Melchiorri, G. Leonardi, C. Albera, P. Scacciatella, P. Pellicori, D. Stolfo, A. Grillo, B. Fabris, R. Bini, A. Giannoni, A. Pepe, L. Ermini, S. Seddone, G. Sinagra, F. Antonini-Canterin, S. Roatta, "Assessment of Phasic Changes of Vascular Size by Automated Edge Tracking-State of the Art and Clinical Perspectives", frontiers in Cardiovascular Medicine, 2022

[6] P. Policastro, G. Chiarion, F. Ponzio, L. Ermini, S. Civera, S. Albani, G. Musumeci, S. Roatta, L. Mesin, "Detection of Inferior Vena Cava in Ultrasound Scans through a Deep Learning Model", Electronics, 2023

[7] P. Policastro, L. Mesin, "Processing Ultrasound Scans of the Inferior Vena Cava: Techniques and Applications", bioengineering, 2023

[8] F. Molinari, corso di Bioimmagini presso Politecnico di Torino, 2021

[9] Z. Zhao, P. Zheng, S. Xu, X. Wu, "Object Detection with Deep Learning: A Review", IEEE transaction on neural networks and learning systems

for publication, 2019

[10] A. Kumar, A. Kalia, H. Pradesh, A. Sharma, "Object Detection: A Comprehensive Review of the State-of-the-Art Methods", International Journal of Next-Generation Computing, 2020

[11] G. Cirrincione, "Advanced Deep Learning and Transformer" course, at Palermo University, 2023.

[12] R. Girshick, J. Donahue, T. Darrell, J. Malik, UC Berkeley, "Rich feature hierarchies for accurate object detection and semantic segmentation", 2014

[13] He, K., Gkioxari, G., Dollar, P., and Girshick, "Mask r-cnn", IEEE International Conference on Computer Vision (ICCV), 2017

[14] K. Li, L. Cao, "A Review of Object Detection Techniques", 2020 5th International Conference on Electromechanical Control Technology and Transportation (ICECTT)

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. C. Berg, "SSD: Single Shot MultiBox Detector", 2016

[16] J. Redmon, S. Divvalay, R. Girshick, A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", 2016

[17] YOLOv8 repository, URL:`https://github.com/ultralytics/ultralytics`

[18] YOLOv8 architecture, URL:`https://github.com/ultralytics/ultralytics/issues/189`

[19] J. Guo, H. Lou, H. Chen, H. Liu, J. Gu, L. Bi, X. Duan, "A new detection algorithm for alien intrusion on highway", Scientific Reports, 2023

[20] T. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, S. Belongie, "Feature Pyramid Networks for Object Detection", 2017

[21] J. Wu, S. Liao, "Traffic Sign Detection Based on SSD Combined with Receptive Field Module and Path Aggregation Network", Computational Intelligence and Neuroscience, 2022

[22] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, A. van den Hengel, "A Survey of Appearance Models in Visual Object Tracking", 2013

[23] N. Wojke, A. Bewley, D. Paulus, "Simple online and realtime tracking with deep association metric", 2017

[24] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft, "Simple online and realtime tracking", 2017

[25] MOTmetrics,URL:`https://github.com/cheind/py-motmetrics.git`

[26] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking", 2016

[27] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, C. Tomasi, "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking", 2016
[28] CVAT, URL: `https://www.cvat.ai/`
[29] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, "Learning Deep Features for Discriminative Localization", 2015
[30] M. Bany, M. M. Yeasin, "Eigen-CAM: Class Activation Map using Principal Components", 2020