POLITECNICO DI TORINO

Master's degree course in Mechatronic Engineering

Master's Degree Thesis

Deep learning computer vision algorithms for apple localization and tracking - Simulation, implementation and validation

Supervisor
prof. Marcello Chiaberge
Co-supervisors
dott. Vittorio Mazzia
dott. Francesco Salvetti

Candidate
Davide Blasutto
ID: 259298

Academic Year 2020-2021

Abstract

The world population is growing faster than ever, with an expectation of 10 billion inhabitants on the planet by 2050. To sustain such rapid growth, the agri-food sector must necessarily improve its production capacity and its efficiency, innovating through technological drivers that aim at optimising and automating the process, while at the same time embracing approaches that can guarantee the sustainability of the supply chain. In particular, automated and precision agriculture is spreading across the industry, both through fully automated machines and through cooperative robots. Computer vision is the main enabler of this industrial shift, thanks to the enormous improvements the field has experienced through Machine Learning and Deep Learning. These technologies radically changed the way the tracking and detection problems are approached, making real-world applications much more convenient and effective. The main industrial applications revolve around localisation of crop fruits, assessing of its maturity state and its agricultural needs, building a 3d map of the orchard and being able to navigate autonomously through real-time mapping of the surrounding environment. For every of the listed applications, the foundation of the system is a reliable model that is able to detect and track an object consistently. The best solutions known in the literature to accomplish such a task are Mask-R CNN, SSD and YOLO for the detection part and SORT and DeepSORT for tracking. This thesis aims to study a system capable of detecting and counting the fruits of a crop through an implementation of Y.O.L.O. V4 combined with a SORT tracker. The main goal of the system is to be reliable, fast and flexible. While the system has been tested on the specific case of apple orchards, it is able to perform counts for any type of fruit. Several steps have been taken to develop and validate the system. First, a simulation environment was built to correctly validate the system results in the detection, tracking and counting parts, in order to have a reliable reference. This environment was developed with real-world counterparts as a reference, also implementing eight different light conditions representing eight different hours of the day. Second, the performance of the system has been validated in real-world scenarios through videos under different conditions. The last part presents a re-evaluation of the previous work, both virtual and real, by retraining the YOLO model with a dataset provided by PIC4SeR (Politecnico of Turin Interdepartmental Centre for Service Robotics) that features

apple orchards located in the countryside near Cuneo. The resulting system shows an average counting error spanning from 7% to 13% both in the simulated and in the real environments, with sensitivity on the measure due to light conditions. After the retraining, the counting error outputs minor improvements in the simulations, while in the real applications it is seeing counting errors spanning from 11% to 6%. This work can be taken as a basis for future developments of more advanced systems, capable of carrying out automatic harvesting, perform crop mapping operations or assess fruit maturity state.

Acknowledgements

Al termine di questo percorso conclusivo, vorrei ringraziare in primis il laboratorio PIC4SeR per l'opportunità offerta con lo sviluppo di questa tesi, nelle persone del Professore Marcello Chiaberge, Vittorio Mazzia, Francesco Salvetti, Simone Cerrato e Simone Angarano.

Si conclude con questo elaborato un capitolo molto importante. Sebbene felici e relativamente spensierati, questi ultimi anni accademici sono stati sfidanti per molte ragioni. In questo spazio desidero dunque ringraziare le persone che mi hanno accompagnato in questi anni e che hanno reso tutto questo possibile. Voglio ringraziare in primis me stesso per aver mantenuto sempre la forza di volontà nel seguire il proprio istinto, i propri desideri ed i propri obiettivi, di non seguire imperativi altrui e di farsi carico e responsabilità delle proprie scelte, nonostante queste siano spesso le più dolorose ed impegnative.

Vorrei poi ringraziare la mia spalla più importante, i miei familiari più stretti. In primis mio padre, mia madre e mia sorella, per avermi sempre dato la forza di volontà di non mollare di fronte alle difficoltà e di credere nelle mie capacità. Dedico ogni secondo di questo traguardo a voi. In particolare, papà, tutto questo è per te.

Ringrazio gli amici più stretti del gruppetto, con i quali ho condiviso momenti memorabili sia dal punto di vista personale che professionale e che mi hanno (quasi) sempre supportato dal punto di vista morale. Sarà un piacere coltivare il nostro rapporto nel futuro, da colleghi ma soprattutto da uomini. In particolare ringrazio Fado, per avermi sopportato nei frequenti momenti di oblio e Manu, per le birre defaticanti del sabato sera.

Un grazie anche al mio caro amico Davide, con il quale da ormai tempo immemore condivido esperienze di vita, assieme ad oneri e onori delle nostre scelte, tra una sessione di studio in CDS e una buona birra.

Un pensiero anche per una persona speciale, presente dall'inizio di questo percorso universitario. Sebbene è stata dura sopportarmi a tratti, grazie per essere stata la mia spalla, una sincera guida, un importante punto fermo e

un grande motivo di crescita. Sei stata parte di me, senza il tuo supporto morale e le tue esortazioni non sarebbe stato possibile arrivare fino qua.

Ringrazio tutti i miei colleghi in SavingBud, in primis Francesca per la disponibilità dimostrata nel farsi carico di lavoro straordinario, per aver portato avanti il corso IVASS ed aver sopportato i miei costanti ritardi mattutini dovuti alle soventi nottate di lavoro; in secundis tutto il team per avermi dato la spinta morale necessaria a concludere questo elaborato.

Ringrazio inoltre i miei coinquilini, Giulia, Piero e Christian per avermi supportato con abbondanti camomille e limoncelli nei momenti di difficoltà dovuti all'elevato stress di questo intenso periodo lavorativo.

Chiudo i ringraziamenti invitando tutti i lettori di questo elaborato a vivere da protagonisti l'esistenza di vita. Non ci deve essere paura di seguire il proprio credo ma anzi, volontà di andare alla scoperta di se stessi attraverso l'ignoto, avendo la forza d'animo di rimanere fedeli alla propria persona, rivelando il percorso e scopo riservato a noi, proprio e diverso per ciascuno. L'unica scelta sicura che merita di essere compiuta è prendersi il rischio di essere noi stessi.

Audentes fortuna iuvat.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The world population is growing faster than ever, with an expectation of 10 billion inhabitants on the planet by 2050. The increase represented by the amount of food produced today and the amount needed to feed the earth's population in 2050 is almost 56%, as there will be about 3 billion more people to feed than there were in 2010 [34]. As median income rises along with life quality, people are expected to increasingly consume more resource-intensive, animal-based foods. At the same time, the agriculture field needs to stop the conversion of remaining forests to agricultural land and cut greenhouse gas emissions from agricultural production. The technological challenge faced by engineering and research is to find ways to improve farming efficiency in every category of agricultural operation (harvesting, sowing and maintenance), while preserving the resources employed and possibly providing for sustainable ways to improve it.
These objectives are the ones tackled by Smart Farming and Precision Agriculture, quickly spreading across the industry, both through fully automated machines and through cooperative robots aimed at automatise and optimise farming management.
The approach proposed by *Precision Agriculture* is to define a decision support system for the whole farm management. The term "precision" indicates how, thanks to the state-of-the-art tools used, it is possible to perform precise interventions, exactly and only where the intervention is needed, at the right time, responding to the specific demands of individual crops. The classical way of dealing with agricultural interventions is based on the operator expertise and spotty checks on the whole field, followed by operations that

regard every piece of land in the same way. Thus, the methods proposed by precision agriculture are providing the farm management with superior levels of precision and therefore efficiency.
To provide such approach to the industry, the technology employed is aimed to collect information in order to make data-driven decisions. An example can be GPS information about the land or drone photos of specific plots of land. In fact, the most used technologies in the sector are Drones and satellite imagery, Internet of things (IoT), smartphone applications, Machine learning and its applications on Robots. Nowadays the research is focusing on further areas as the precise monitoring of the field and crop in order to optimise the use of irrigation, fertiliser and pesticide, the use of spectral cameras on fruit to detect damages or infections and to determine the actual ripening and an ever more automatic process of harvesting. The process of automation combined with world perception with cameras, is the topic of interest of this work. Machine learning and computer vision is the main enabler of the automation, thanks to the enormous improvements the field has experienced in recent years through the introduction of Deep Learning. The latter have radically changed the way the tracking and detection problems are approached, making real world applications much more convenient and effective.

Figure 1.1. Examples of harvesting automation operated with various types of robots on strawberry fields

The main industrial applications revolve around localisation of crop fruits, assessing of its maturity state and its agricultural needs, building a 3d map of the orchard and being able to navigate autonomously through real-time mapping of the surrounding environment. For every of the listed applications, the foundation of the system is a reliable model that is able to detect and track an object consistently. This thesis work responds to this need by

detailing the construction, simulation and subsequent test of a state-of-the-art system able to detect, track and count apples, representing one of the few iterations of such a system currently recorded on literature when applied to fruit counting.

1.2 Objectives

Scope and objective of the thesis work is build a state-of-the-art system able to assess the fruit count from a video of an apple crop. The scope of this work is:

- Research and assess the current state of the art for the proposed problem of counting apple or crop fruits;
- Build a system able to detect and track with sufficient precision every apple of the crop;
- Validate the system built by using a custom virtual environment that resembles a real world scenario, in different working conditions;
- Validate the system built by testing it on some real-world videos, in different working conditions;
- Compare the results obtained after retraining the network on a custom apple dataset;

1.3 State of the art

Many studies have been proposed in the recent years featuring systems devoted to fruit counting, with various approaches both for the detection and the tracking part. Roy and Isler [21] conducted an extensive comparative of fruit detection and counting methods for yield mapping, proposing a semantic segmentation-based approach. Combining a semi-supervised method featuring a Gaussian Mixture Model with the deep learning-based methods, they achieved yield estimation accuracy ranging from 95.56% to 97.83%.
Liu et al. [17] showed a system featuring a FCN-based segmentation and tracking based on the Hungarian Algorithm where the objective cost is determined from a Kalman Filter corrected Kanade-Lucas-Tomasi (KLT) Tracker. The count is further refined using Structure from Motion (SfM) algorithm to calculate relative 3D locations and size estimates to reject outliers and

double counted fruit tracks, achieving precision ranging from 93% to 97%.
Koirala et al. [22] performed real time mango fruit detection with a Mango-YOLO model showing an F1 score of 0.968, AP of 98.3% and an inference speed of 14 FPS on a NVIDIA GeForce GTX 1070 Ti GPU, using as tracker a system composed of Kalman filter and Hungarian algorithm.
Other applications involve depth measure through an appropriate RGB-D camera, as proposed by Xue et al. [25]. The system involved multiple scale faster region-based convolutional neural networks (MS-FRCNN), using colour and depth images acquired with an RGB-D camera. The system obtained a recall, precision and F1 score of respectively 0.962, 0.931 and 0.946.
Jarvinen et al. [15] developed a detection and counting system for counting fruit on apple trees using an object detector based on the Faster R-CNN architecture and optical flow for tracking, obtaining a precision of 92% and recall of 82%.
Chen et al. [13] proposed a blob detector based on a fully convolutional network and a counting algorithm based on a second convolutional network that estimates the number of fruit in each region. Linear regression model maps fruit count estimate to a final fruit count.
As for the most recent works based on YOLO detection network, the vast majority of the literature is devoted only to the detection part. Liu et al. [27] and Lawal [27] proposed a YOLOv3-based tomato detection systems that showed AP values of 96.4% and 99.5%, respectively, showing inference time around 19 FPS using an NVIDIA GeForce GTX 1070 Ti GPU and an NVIDIA Quadro M4000 GPU.
Li et al. [28] developed Lemon-YOLO for detecting lemon fruits, replacing Darknet-53 with an ResGNet34 network. The system showed an AP of 96.28% and a detection speed of 106 FPS with a Tesla V100 GPU.
It is important to note that only one study combined YOLO with a multiple object tracking algorithm for counting fruits. Itakura et al. [26] used YOLOv2 and Kalman filter to count pear fruits from a video to achieve an AP of 97% in detection and an F1 score in counting of 0.972.

1.4 Proposed solution

The proposed solution is an integration of the latest and most up to date technology available at the moment of writing, in terms of detection and tracking. This kind of setup, represented by a counting system made of a detector and a tracker, is representing one of the few iterations currently recorded on literature when applied to fruit counting. Another point of innovation of the present work is represented by the objective to operate a vast validation process of the system in different working conditions with the help of a simulated environment, looking for performance variance. In the system, the video is fed through a CNN network that at first performs the detection of the defined object (in this specific work, an apple is chosen but it can be arbitrarily chosen), then every object is tracked with a proper module and at the final stage is every unique track ID is counted. The detection part is carried over with a YOLOv4 neural network, which is the state of the art for the computer vision detection, while the tracking part features for a SORT tracker, chosen for its state-of-the-art speed and performance. To validate the system, the steps taken are:

- Build a custom virtual environment that can work as ground truth reference for the detector and tracker. Such system feature 8 different lightning conditions that are simulating various working hours;
- Compare the obtained results with real world videos, whose ground truth have been obtained through manual counts;
- Retrain the detection neural network on a custom apple dataset, directly taken on-site in a orchard outside Cuneo;

Chapter 2

Background

This chapter introduces the theoretical concepts that are covering the background of the present thesis work. A brief review is given about Machine Learning, Deep Learning and how these models are used into the field commonly recognised as *Artificial Intelligence*. The following paragraphs are revolving around *Convolutional Neural Networks* and their application in Computer Vision applications. A further theoretical introduction is given for the *Multiple Object Tracking (MOT)* problem, followed along with the main metrics used to evaluate the present work. The last paragraph is devoted to briefly describe the software infrastructure used in the development of the project.

2.1 Artificial intelligence

The word "Artificial Intelligence" is commonly used nowadays in a broad band of topics. The common meaning is referred to "a machine able to make autonomous choices, while learning from its mistakes". While the word was first used in 1959 by A. Samuel [2], the recent development in Machine Learning and Deep Learning modelling allowed for major field development in recent years. The field started with researchers thinking about building an electronic brain. In 1943, Warren McCullouck and Walter Pitt proposed the first work recognised as AI: a model of artificial neuron, that could distinguish two different categories of inputs by testing whether a certain linear function f(x,w) is positive or negative, but the weights had to be set correctly by a human operator. In the 50s, John McCarthy, Marvin Minsky, Claude Shannon, Nathaniel Rochester, Arthur Samuel, Allen Newell and Herbert Simon implemented a program able to learn checkers strategies, while in 1957,

Frank Rosenblatt invented the Perceptron, the ancestor of neural networks, made by only one layer and just enough to learn the weights of a linear function without human action. The research on the topic have been stalling until the 90s, where the increasing computational power matched the ongoing scientific discoveries in various fields such as robotics, medicine, mathematics, physics and economics. The 11-th May 1997 is pointed as an historical date since Deep Blue, the IBM supercomputer, became the first computer chess-playing system to beat the world chess champion, Garry Kasparov, demonstrating that machines can beat the human in tasks were intelligence is employed.

Figure 2.1. World Chess Champion Garry Kasparov playing his fourth game against the IBM Deep Blue chess computer

In 1998,Yann LeCun invented LeNet-5 [4], one of the earliest convolutional neural networks that promoted the development of Deep Learning, trained on the MNIST dataset for hand-written text recognition. Moreover, the capabilities of AI systems have been proven in public competitions such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where AlexNet was presented in 2012, becoming the milestone among Convolutional Neural Network (CNN) for the time. In the same category we can remember the famous GoogLeNet, ResNet and YOLO. In 2011, the IBM's question answering system called *Watson*, defeated the two former champions in the game *Jeopardy!*, while in March 2016 AlphaGo won 4 out of 5 games of Go in a round against the champion Lee Sedol, and the next year the same computer program won a three-game match with Ke Jie, ranked number one in the

world in Japan Go Association's. Until recent, many other applications are currently used on a daily basis by digital users, such as "Chatbots", "Search and Recommendation" algorithms used in search engines and "Text Editors or Autocorrect" used my most smartphones.

2.1.1 Machine Learning

Machine Learning is one of the main pillars of modern Artificial Intelligence. It is defined by A. Samuel in 1959 [2], as:

> *"the field of study that gives computers the ability to learn without being explicitly programmed."*

In 1997, T. Mitchell gave a broader insight [3]:

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E"*

The field of applications of such models are ranging from medicine, to speech recognition, to computer vision and in general wherever the tasks are such that a direct algorithm cannot reach a satisfactory performance. Commonly used tasks are usually classification or regression, clustering, anomaly detection, dimensional reduction or predictive analysis.

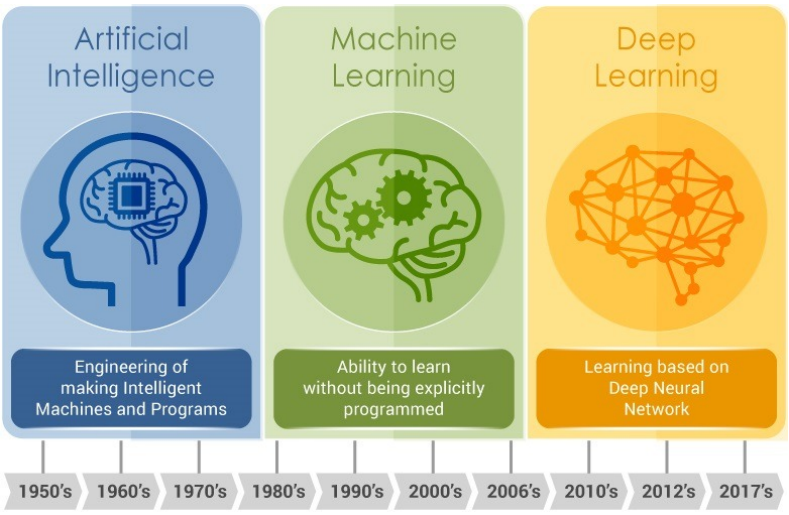


Figure 2.2. Visual representation of the evolution between Artificial Intelligence, Machine Learning and Deep Learning, placing them in a timeline

One of the first industrial application of Machine Learning is dating to 1990s, and is represented by the email spam filter. After that, many other applications have widely spread into millions of lives.

The main Machine Learning Systems can be classified according to the amount and type of supervision they get during training.

- **Supervised learning**, where the training set fed to the algorithm include the desired solutions, called "labels". For example, predictors for housing prices or classificators for spam emails belongs to this category;

- **Unsupervised learning**, where the training data is unlabeled, thus does not contain info about the desired output. The scope of the Machine Learning system in such a systems is clustering, anomaly detection or dimensionality reduction. An example for the latter could be a model able to group a website visitor list, outputting "clusters" where every member inside have characteristics in common. Another example is "anomaly detection", where a Machine Learning model is able to detect outliers from a defined set of data;

- **Semisupervised learning**, where unlabeled data is employed along with labeled data, in order to produce a considerable improvement in learning accuracy. A common example of this category is cloud storage photo services, where the systems usually automatically clusters a certain amount of people appearing in the pictures the user uploaded. Once inserted the name of the persons, the model is able to recognise the persons in every other photo;

- **Reinforcement learning**, where the learning system called "agent" take actions in a given environment in order to maximise its reward, while the task of the system is to find the best strategy called "policy". This types of models are used in fields like game theory, control theory, operation research, information theory. DeepMind's AlphaGo, is an example of this kind of systems. In 2017, the model won a game of the game "Go" against Ke Jie, the former world champion. The system applied the policy learned in the thousands of games played during training phase;

2.2 Deep Learning

2.2.1 History

Deep learning is a subset of Machine Learning, which is in turn a sub-field of Artificial Intelligence. In 2015, Yann LeCun described Deep Learning as [7]:

> *"Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. [. . .] The key aspect of deep learning is that these layers are not designed by human engineers: they are learned from data using a general-purpose learning procedure"*

In literature, Deep Learning belongs to the family of Artificial Neural Network called *ANN*, but in most cases the two terms can be used interchangeably. ANNs take inspiration from biological human brain architecture and how they interact with neurons, but not trying to be realistic models of the brain.
The first neural network model is dated 1943, from McCulloch and Pitts [1], being the network a binary classifier able to recognise two different labels while still required some weights to be tuned by hand from an operator. In 1959 by Arthur Samuel, an IBM employee who developed an algorithm able to play checkers. His program was able to improve its performance by learning from past moves, combining them in a specific reward function and estimating the best move by using a so-called minimax strategy, which eventually evolved in the famous *minimax algorithm*. In 1957, Frank Rosenblatt combined Donald Hebb's model of brain cell interaction with Arthur Samuel's Machine Learning efforts and created the *Perceptron*. The latter initially planned as a machine, evolved in a program for the IBM 704 for image recognition implementing a single layer able to learn weights from a linear function. The Perceptron was based on the Stochastic Gradient Descent (SGD), still used today in the learning process of deep neural networks. In the 1960s, it was discovered that providing and using two or more layers in the perceptron offered significantly more processing power than the single layer, introducing the use of multi-layers and opening a new path in neural network research. In 1969, Marvin Minsky along with Seymor Papert, published a research that demonstrated the inability for a Perceptron to learn the trivial exclusive-or function (called *"XOR problem"*), even with unlimited

training time. Being the XOR Problem non-linear and the Perceptron a linear model, the latter was not able to correctly predict the system. This event blocked substantially further research into the field of both Machine Learning and AI, in what has later been called "Dark age or AI winter". In 1986, Geoffrey Hinton along with Ronald Williams and David Rumelhart, in 1986, turned the doom from machine learning with the famous publication entitled "Learning representations by back-propagation errors", giving a proof that neural networks with multiple hidden layers could be quickly trained. Combining with non-linear activation functions made the models able to solve non-linear problems. In particular they demonstrated that neural networks are universal approximators, systems able to approximate any continuous function. This was done by means of a new algorithm called *backpropagation* that still today represents a major pillar of Machine Learning.

The big bottleneck for AI and Neural Networks for that age was the computational requirements that made any network with more than two layer, too complex to be calculated with the devices available at that time, resulting in another period where neural networks were put aside for a second time. These limits are widely bypassed by the modern hardware capabilities made available by CPU and GPU computing.

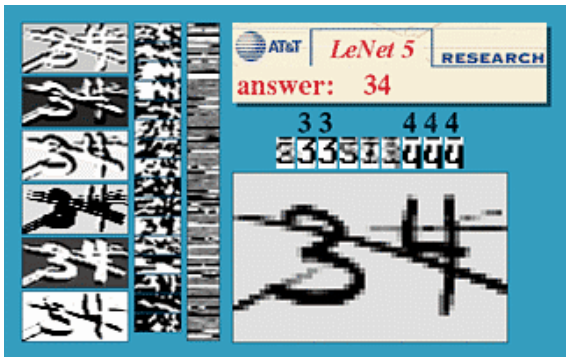


Figure 2.3. A demonstration of *LeNet-5* by Yann LeCun in 1995, one of the first neural networks able to perform hand-written text recognition

Neural Networks started to regain interest in the research area with the introduction in 2006 of unsupervised pre-training for networks. With unsupervised learning, a neural network is fed with unlabelled data and set up the learning process to look for recurring patterns. Researchers could now using this methodology to train more complex networks with multiple hidden layers, giving the birth to field known as *Deep Learning*.

2.2.2 Artificial neuron or Threshold Logic Unit (TLU)

The basic element on which a neural network is based is represented by the Threshold Logic Unit or (TLU).

It can be represented as a system able to take several inputs x_n and produce an output y, composed of the following elements:

- an input, x_i
- a weight, w_i
- a bias, b
- an activation function, f
- a threshold related to the activation function, θ
- a output function, y

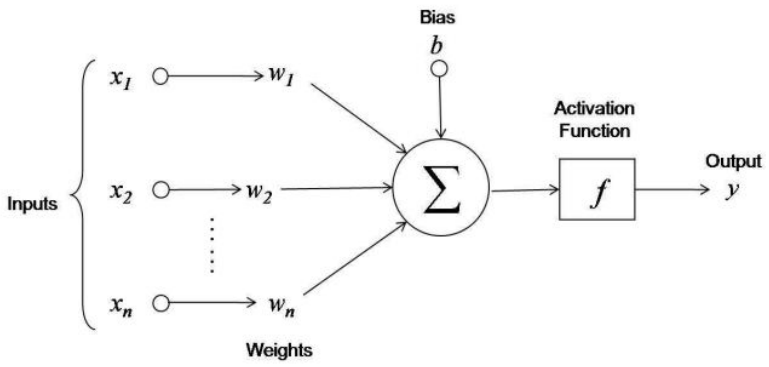


Figure 2.4. Artificial Neuron or Threshold Logic Unit (TLU) structure

The inputs are representing the information to be treated by the neural network, that can be any data able to be decomposed into a digital information. Example of this can be images, text information or sound clips. The TLU combines linearly the weighted inputs, while adding the bias b. The latter is passed to the activation function f, that is activated only if the latter is greater than the threshold θ. The result is then passed to the output function, that is responsible for the final data output y. The equation for the output of the neuron will be then:

$$f = f(w_1x_1 + w_2x_2 + ... + w_ix_i + b) \tag{2.1}$$

2.2.3 General architecture

The term *Artificial Neural Networks*, called in short *ANN*, is used to identify the mathematical structure of a deep learning model. An ANN is based on a collection of the basic units represented by the artificial neuron or TLU, connected each other while sending and receiving signals.

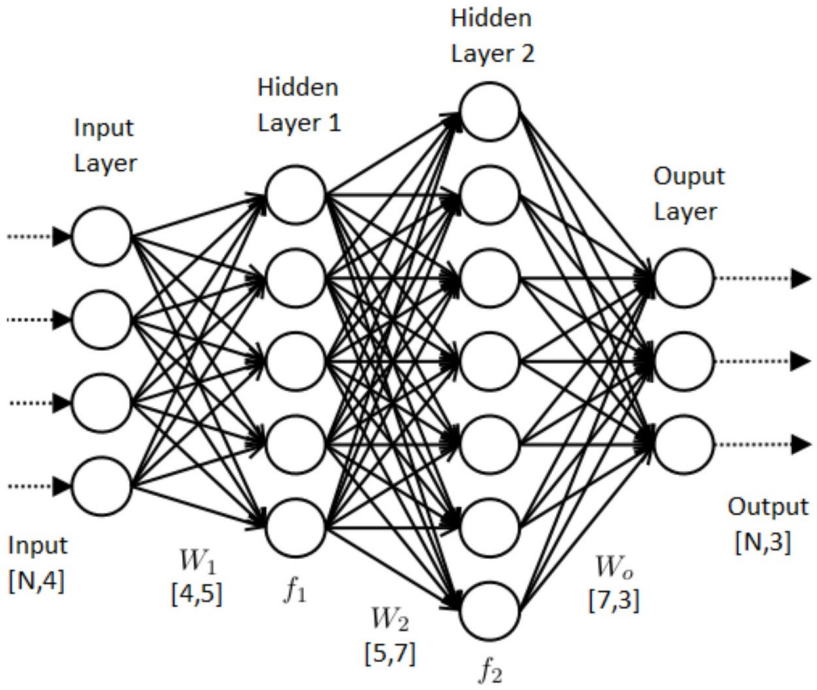


Figure 2.5. Artificial Neural Network general structure

The neurons are organised in various layers whose number and neuron count can vary basing on the computation needs. The first layer is the so-called *input layer* and is directly connected with the input data. The last one is called *output layer* and represent the last step that produces the network's final output. In between the two layers, there are the so-called *hidden layers* that produce the signal able to generate the output signal. The presence of hidden layers and non-linear activation functions is what makes the network able to be universal approximators, able to approximate any function. Every neuron in each layer has its own weight w_i associated to it, representing the neuron contribution of the synapse. The learning process is aimed to tune every neuron weight in order the obtain the wanted result represented by the training data. The layers are linked through connections that, in the most simple ANN example, link every neuron of two layers to each other.

2.2.4 Activation functions

Activation functions f are fundamental in both the performance such as linearty, continuity, derivability and simplicity of the network, while also improving widely its learning processes. Several activation functions are used in literature. The activation function of a node defines the output of that node, given an input or a set of inputs. A standard activation function can be the digital switch which can be set to "ON" or "OFF" and whose output changes in a non-continuous fashion. In order to solve nontrivial problems however, nonlinear activation functions are needed. Some of the various activations functions used are presented in the paragraphs below.

Heaviside

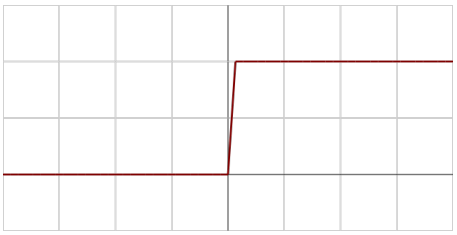


Figure 2.6. Heaviside function or Step activation

It is the first function used for neuron activation purposes and it resembles the step function that can be seen above. The function is represented by:

$$f(x) : \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{2.2}$$

As it can be seen, its output can only be 0 or 1 and the turning point is discontinuous. This creates unwanted non-linearities and errors when the input values is near the turning point. For this reason, this type of function has proven to be unreliable for learning purposes, since a slight variation of the weights can easily cause a switch in the output from a state to another.

Linear

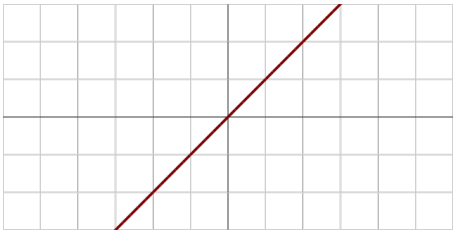


Figure 2.7. Linear function

The second function presented is the linear ramp, which provides a smoother state transition than before while being continuous, but not derivable in two points. The function can be represented by the following formula:

$$f(x) = cx \tag{2.3}$$

Sigmoid

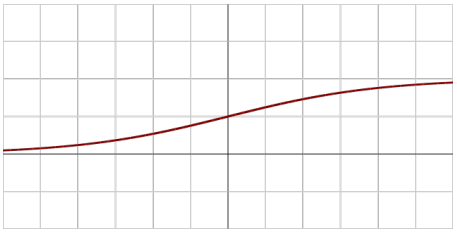


Figure 2.8. Logistic sigmoid function

To make the output state transition even smoother, the sigmoid function has been introduced. It is continuous and provides a smooth state transition. It is one of the most common functions used in Machine Learning nowadays.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

Hyperbolic tangent

The hyperbolic tangent is similar in shape to a sigmoid function, and is represented by the following function:

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

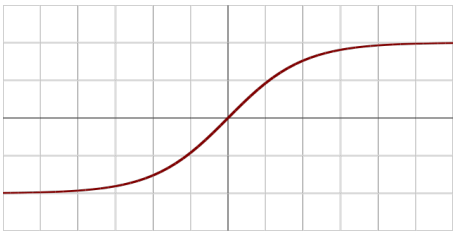


Figure 2.9. Hyperbolic tangent function

This function has the special characteristic of producing an output that can also be negative. This function is used to avoid the phenomena of neuron saturation, where the hidden layers assume values close to boundaries values and the output nodes output of the neuron is saturated to its minimum or maximum value.

ReLU

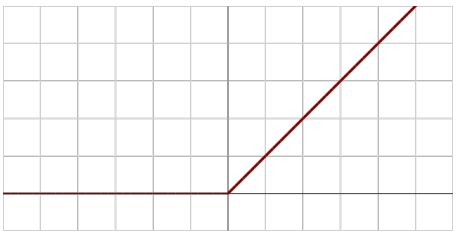


Figure 2.10. ReLU function

The ReLU function has the following form:

$$f(x) = max\{0, x\} \tag{2.6}$$

Its main characteristic is that while being very simple to implement due to its piece-wise structure of a ramp imposed to a "0" function, it remains non-linear so its suited to be implemented in complex neural networks. It is one of the most used activation functions in modern applications of Machine Learning.

2.2.5 Gradient Descent

Like previously explained, with every neuron is associated a specific weight w_i and bias b_i. The scope of the learning process is to find the weights and biases for every neuron in order for the neural network to give the desired output. The learning process is based upon a cost function C, to be minimised iteratively over various steps called *epochs*. Following the definition of Mean Square Error (MSE), the structure of the cost function can be defined as follows:

$$C(w,b) = \frac{1}{2n}\sum_x ||\hat{y} - y(x)||^2 \tag{2.7}$$

While w and b are representing weights and biases of the networks, n represents the total amount of neurons in the network. Moreover,
hat(y) represent the desired output while $y(x)$ represent the actual output of the neural network. The goal of the learning process is to minimise the cost function $C(w,b)$, thus the problem can be framed into a minimisation problem. Since the dimension of the problem is too big to be addressed in a analytical fashion, the general Gradient Descent Algorithm is used. The goal is to find a global minima of the cost function $C(w,b)$. For a small variation of the cost function, a generic n-dimensional input v is considered. The increment of every contribution v_i can be derived as follows:

$$\delta C = \frac{\partial C}{\partial v_1}\triangle v_1 + \frac{\partial C}{\partial v_2}\triangle v_2 + ... + \frac{\partial C}{\partial v_i}\triangle v_i + ... + \frac{\partial C}{\partial v_n}\triangle v_n \tag{2.8}$$

Which can also be written exploiting the gradient definition, as follows:

$$\nabla C = (\frac{\partial C}{\partial v_1}, ..., \frac{\partial C}{\partial v_n})^T \rightarrow \delta C \approx \nabla C \delta v \tag{2.9}$$

The goal for the learning algorithm is to find the set of appropriate variables v_i that make ΔC negative. Since the latter is minimised, the scope is to find the global minima of $C(w,b)$. Since the process is iterative as stated before, a parameter called *learning rate* is defined. The algorithm objective is find the variation Δv that minimises the cost function, towards its global minima.

$$\Delta v = v' - v = -\eta \nabla C \tag{2.10}$$

This value represent how much variables are increased for each learning step. At this point, the gradient descent equation can be obtained by:

$$\Delta C = -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2 \tag{2.11}$$

Thus, the new set of inputs is:

$$v' = v - \eta \nabla C \tag{2.12}$$

The learning rate is crucial for the learning process in order to guarantee a good approximation of ∇C while go towards its global minimum. Often the learning step is variable, so when the global minimum is still far, the process is not slowed down too much by a small learning rate. The latter is reduced when approaching the minima.

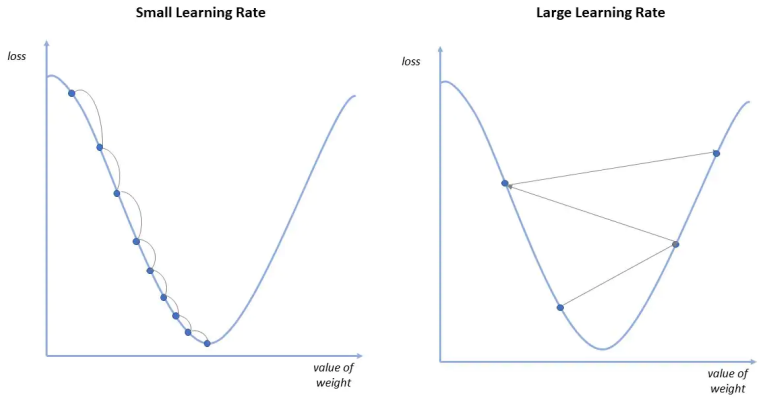


Figure 2.11. Approach towards the minima of the cost function with learning rate modification

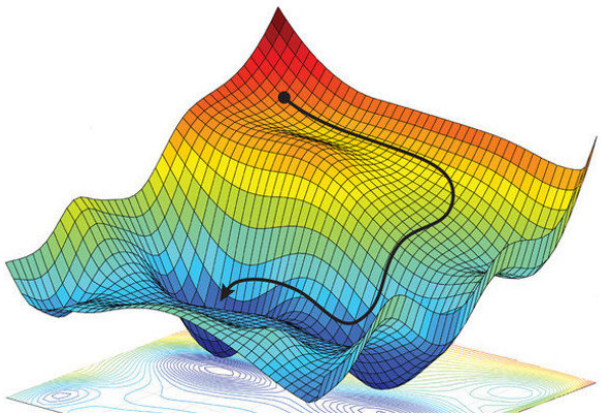


Figure 2.12. Gradient descent process visualisation in a 3d cost function

2.3 Convolutional Neural Networks

In the previous sections, a fully connected network has been evaluated, meaning that every neuron in a specified layer is connected to every neuron in the previous and in the next layer. When dealing with images however, having a fully connected network introduces a wide computational burden that makes the system perform poorly when treating images.

The input layer is connected to an hidden layer but, each neuron is linked only to a specific group of pixels in the input layer. This little window is known as "local receptive field". Each connection of the receptive field features a weight and a single bias. This architecture allows the network to focus on low-level features in the first hidden layer and then assemble the information into a larger higher-level feature in the next hidden layer, building the "convolutional" part of the network. In order to compose the whole hidden layer, the window of the receptive field slides by a quantity called *stride* through the input image pixels creating step by step a layer of hidden neurons. All the pixels in the receptive field share the same neuron, thus share the same weight, vastly simplifying the amount of parameters of the model. The union between all groups of shared variables, weights and biases, is known as *kernel* or *filter*.

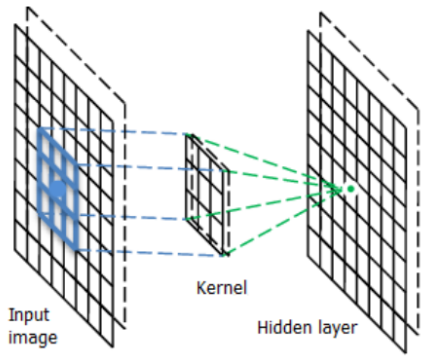


Figure 2.13. Receptive field and its link with the hidden layer

The equation of convoloution is now defined, having σ as a generic activation function related to the neuron and with a $a(x, y)$ the input activation function at position (x, y). The output produced by the $(j, k) - th$ hidden

neuron will be:

$$out_{j,k} = \sigma\left(b + \sum_{l}\sum_{m} w_{(l,m)} a_{(l+j,m+k)}\right) \tag{2.13}$$

As it is immediate to see, since the weights are shared, the number of total parameters used in the network is dramatically reduced concerning the fully connected architecture allowing a faster training process. Filters can effectively seen as masks that ignore everything in their receptive fields except for the specific shape defined by the mask. Examples of filters can be vertical lines or horizontal lines. An hidden layer that use the same filter, is outputting a filter map, enhances the areas in the image that are being activated by the filter. The implementation of Convolutional filters sees them stacked into multiple trainable layers, so every pixel is associated to a neuron in each feature map, and every neuron in a feature map shares the same parameters, thus reducing widely the amount of parameters in the model.

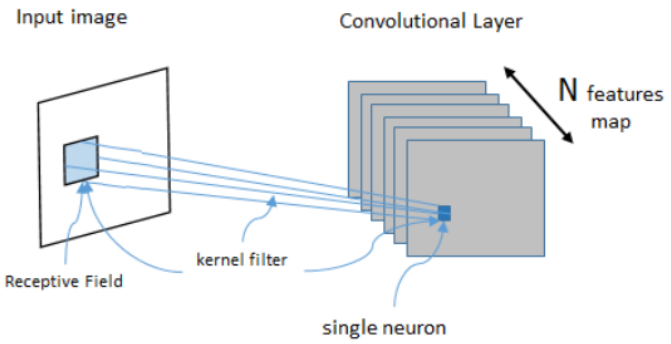


Figure 2.14. Example of feature map in a Convolutional Neural Network

Better control on the output size of the layer can be obtained by setting to 0 the pixels along the image border. A so-called *zero padding* is added, which adds zeros around the last layer in order to have it of the same dimension compared to the previous layer, thus making sure that no information is lost.

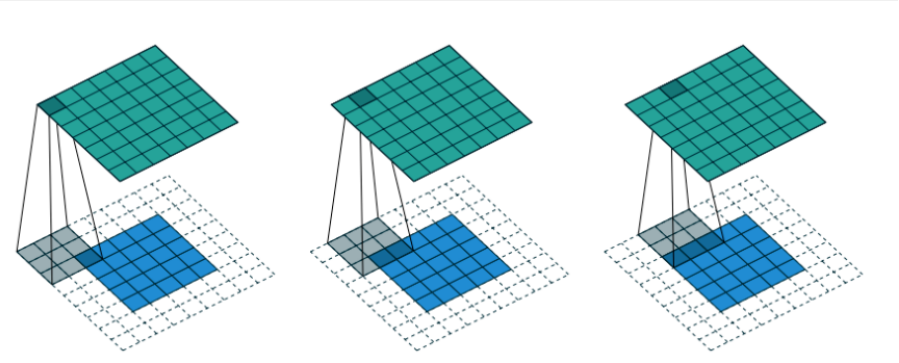


Figure 2.15. Example of a zero padding equal to two pixels

2.4 Computer Vision applications

Computer vision is an interdisciplinary scientific field that aims to gain high-level information from digital images or videos. The recent technological development of model such as CNNs has become the standard in the field, allowing machines first to gain meaningful data from the video or image and secondly, allowing the machines to derive further information from the environment or take make autonomous decisions such as navigate autonomously. One of the most common applications for the file is object detection. Object detection is the computer technology that is able to detect instances of semantic objects of a certain class in digital images and videos [5]. The aim is to locate and classify objects in a image or video, associating also a confidence on the prediction. The pipeline for a standard object detection model is divided into three main stages:

- **Region selection**, where the model adopts a multi-scale sliding window on the image, looking for the areas where its most likely to find objects on the image itself. It is quite computationally expensive due to the high number of windows created;
- **Feature extraction**, where the network recognises the different objects by extracting meaningful features from the previous areas;
- **Classification**, where every box found is linked to a precise category or label;

In general, the state of the art object detection models are classified into two categories:

- **Region proposal based**, where a region of the image will be selected and it will be classified to the different categories. The most popular models for this category are *R-CNN*, *Fast R-CNN* and *Faster R-CNN* models.
- **Regression/classification based**, where the process of feature extraction and classification is done only in one step. Models such as *SSD, Single Shot MultiBox Detector* and *YOLO, You Only Look Once* represent the state of the art of the category.

The region-based models, however, showed mediocre performance improvements over time, mainly due to its heavy computational expense derived from the implementation of multiple sliding windows. Regression based framework, where the classification problem is carried out in a single step, have shown much better performance and scalability compared to the first category, making it the effective state of the art for Object Detection, both for speed and accuracy. *SSD* and *YOLO* basic structure are covered into detail in the following sections.

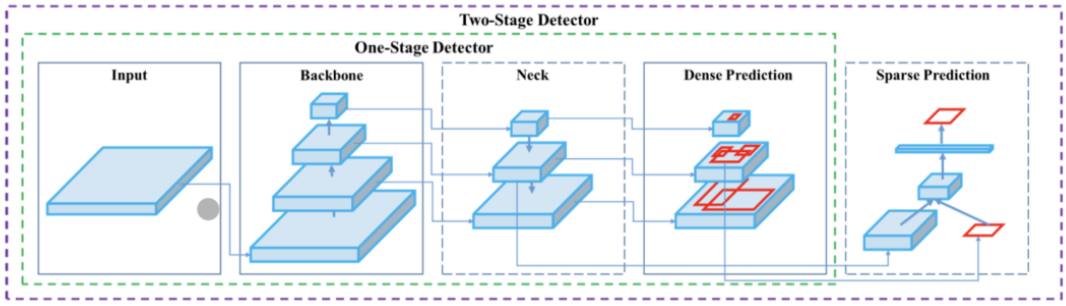


Figure 2.16. Architecture of a one-stage detector compared to a two-stage detector

2.4.1 SSD

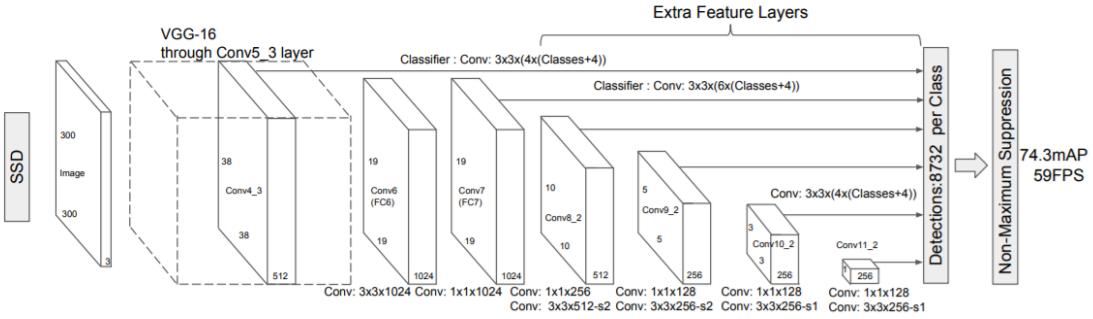


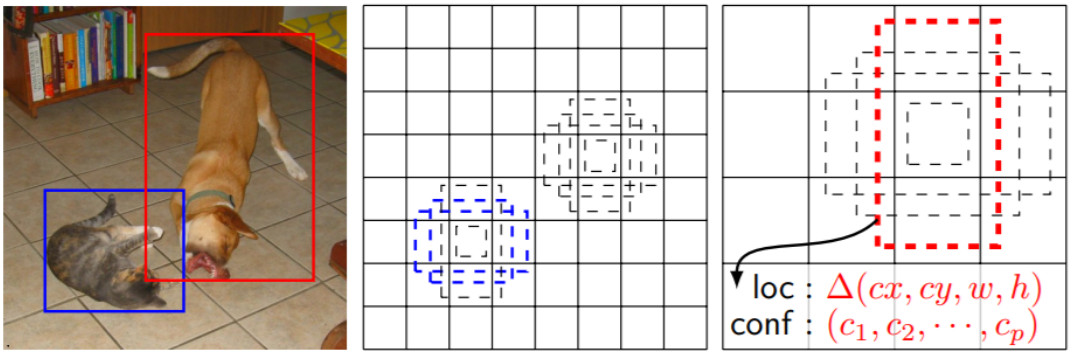
Figure 2.17. Schematic representation of the architecture of the SSD Network

Single Shot multibox Detector is a model that discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. The network also combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes [8]. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification, called base network, in this case represented by VGG-16 [8]. Additional auxiliary structure is added to the network in order to produce detections with the following key features:

- **Multi-scale feature maps for detection**, allowing for predictions at multiple scales by introducing convolutional feature layers to the end of the truncated base network;

- **Convolutional predictors for detection**, each added feature layer added feature layer can produce a fixed set of detection predictions using a set of convolutional filters, indicated on top of the SSD network architecture in Fig. 2. For a feature layer of size m $\times$ n with p channels, the basic element for predicting parameters of a potential detection is a 3 $\times$ 3 $\times$ p small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. At each of the m $\times$ n locations where the kernel is applied, it produces an output value [8];

- **Default boxes and aspect ratios**, associating a set of default bounding boxes with each feature map cell, for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed. At each feature map cell, offsets are predicted relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, c class scores are computed and the 4 offsets relative to the original default box shape. This results in a total of (c + 4)k filters that are applied around each location in the feature map, yielding (c + 4)kmn outputs for a m × n feature map [8];



(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

Figure 2.18. Example of results obtained from SSD framework

2.4.2 YOLO family

YOLO is a object detection architecture proposed by J. Redmond et al. in 2015 [9]. Later in the years it have been progressively improved extending the family to YOLOv2 [12], YOLOv3 [18] and YOLO v4 [24]. It is a fast and accurate object detector, based on the objective of maximising performance over mean average precision (mAP). Each of the YOLO iterations have the following in common:

- **Backbone**, responsible for feature extraction, usually ResNet, VGG, CSPDarknet53 or EfficientNet;
- **Neck**, responsible for collecting feature maps from different stages of the network;
- **Head**, responsible for dense prediction, meaning the prediction of bounding box location and the confidence score associated to it.

The main advantage of one stage detectors such as YOLO is the speed, making them able to work in real-time conditions.

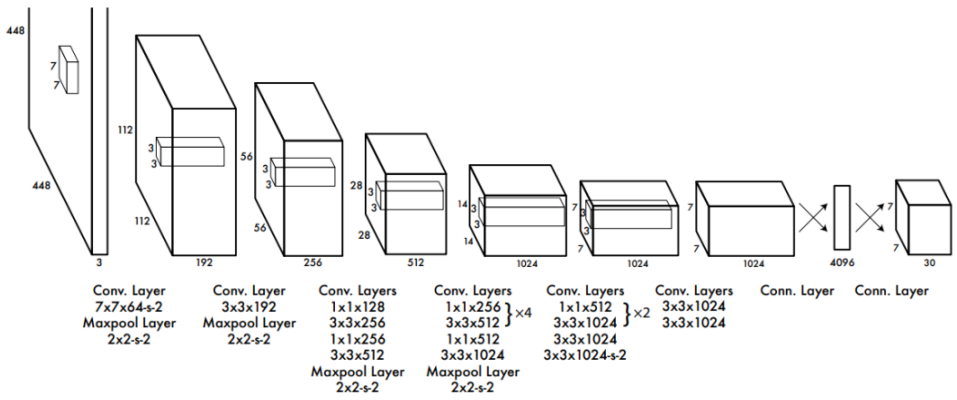


Figure 2.19. The architecture of the first YOLO network

The YOLOv3 algorithm first separates an image into a grid. Each grid cell predicts some number of boundary boxes (sometimes referred to as anchor boxes) around objects that score highly with the aforementioned predefined classes. Each boundary box has a respective confidence score of how accurate it assumes that prediction should be and detects only one object per bounding box. The boundary boxes are generated by clustering the dimensions of the

ground truth boxes from the original dataset to find the most common shapes and sizes.

2.4.3 YOLOv3

In 2018 J. Redmon and A. Farhadi released "YOLOv3: An Incremental Improvement" [18]. It represent a big enhancement with respect from the previous models in terms of accuracy, speed and specificity of the classes compared to previous versions. The backbone is represented by a new net called Darknet-53 made by the creators, featuring 53 convolutional layers. This backbone is on average 1.5 times faster than ResNet101.

While avoiding fully connected layers and pooling layers, the model features logistic classifiers and activations. The performance obtained on the COCO dataset can be seen in figure 2.20, with a mAP of 28.2 and an inference time of 22 milliseconds, almost three time faster than the SSD object detector with the same accuracy.

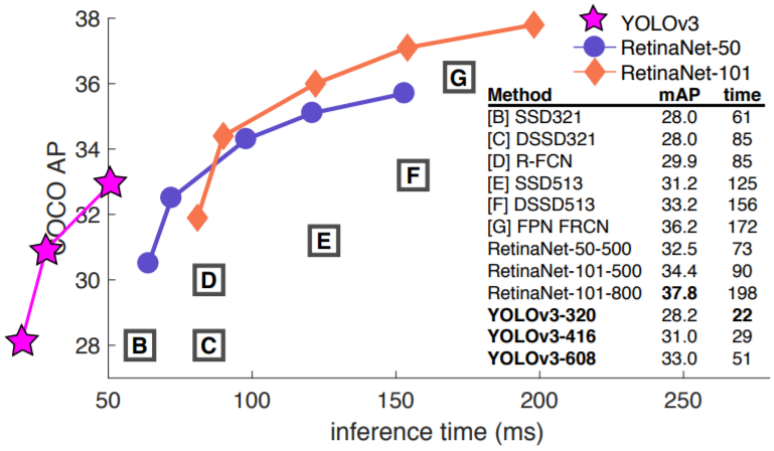


Figure 2.20. Comparison between inference time and MS COCO AP

2.4.4 YOLOv4

On the contrast of the other YOLO iterations, YOLOv4 was the first model not released by J. Redmon but instead released by A. Bochkovskiy, Chien-Yao Wang and Hong-Yuan Mark Liao, in the paper "YOLOv4: Optimal Speed and Accuracy of Obect Detection" [24]. The model was able to reach 43.5 %

AP, running at 65 FPS on a Tesla V100 in MS COCO dataset, beating the previous models as can be seen in figure 2.21.

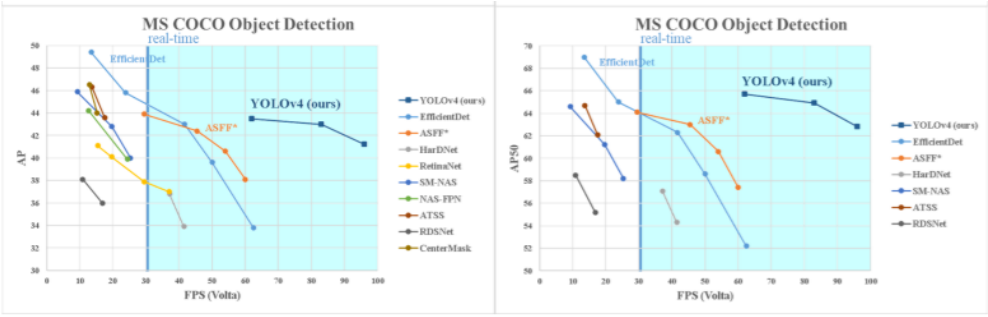


Figure 2.21. Comparison of the detector on NVIDIA Volta GPU Architecture

The network uses CSPDarknet53 as backbone [23], SPP [6] and PAN [16] as neck and YOLOv3's [18] head. The changes introduced in order to enhance the performance from the previous *YOLOv3*, include a so-called "Bag of freebies (BOF)" and "Bag of specials (BoS)". The first are methods that can make the object detector receive better accuracy without increasing the inference cost. These methods only change the training strategy or only increase the training cost [24]. On the model implementation, the following have been introduced:

- **Data augmentation techniques**, such as random erase (select rectangle regions in an image and erases its pixels with random values), CutOut (randomly masks out square regions of input), Mixup (random superimposition of images with different methods) and Cutmix (cutting and mixing images);
- **Regularization**, methods such as DropOut, DropConnect and DropBlock;
- **Bounding box regression loss**, various types of bounding box regression types, such as MSE, IoU, CIoU, DIoU;
- **Normalization**, introduced cross mini-batch normalization (CmBN).

As for the "Bag of specials (BoS)", they represent modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection [24]. Such methods include:

- **Spatial attention modules (SAM)** [19], generating feature maps by utilising inter-spatial feature relationship;
- **Non-max suppression (NMS)**, reducing false positive and false negatives in the case of multiple grouped bounding boxes as prediction;
- **Non-linear activation functions**, such as ReLU, LReLU, PReLU ReLU6, Leaky ReLU, Swish and Mish;
- **Skip-connections**, like weighted residual connections (WRC) or cross-stage partial connections (CSP).

2.5 The Multiple Object Tracking (MOT) Problem

Object detection is the first step required for the purpose of this project that must be followed by object tracking. While the Object Detector recognises and categorises objects in a defined frame, the object tracker scope is track objects from a frame to another, assigning and identity to the object itself. The problem of the fruit counting in a orchard can be framed into the "Multiple Object Tracking (MOT)" problem. While the "Single Object Tracking (SOT)" problem deals with design appearance models and motion models to deal with object scale variations or illumination changes, the MOT problems also sees a wide importance in maintaining the object identity both in intra-object occlusions, initialisation and termination of tracks with similar appearance. Several MOT algorithms have been developed and studied in literature. These can be usually divided into four main categories, based on how tracks are initialised and how the tracks are updated by the algorithm.

- **Detection-based tracking (DBT)** algorithms, initialising tracks based on information coming from a previous detection stage. For every frame of the video, an object detector is employed in order to obtain objects location. The main issue behind these types of trackers is that the final result depends widely on the detector performance, while also requiring the detector to run on every frame, vastly augmenting the computation time for the video;
- **Detection-free tracking (DFT)** algorithms, where is required a manual initialisation of the objects to be tracked. The algorithms are then able to provide the positions of the selected objects in the successive

frames. The main drawback of this approach is that new objects coming into the scene cannot be detected or tracked;

- **Online tracking** algorithms, that rely on the past information available up to the frame, meaning that the algorithm are predicting the future location of the objects;

- **Offline tracking** algorithms, working on every frame of the video in a post-process fashion, in order to derive the final result;

The present project requires the implementation of a online, detection-based tracking algorithm. As for the state of the art of real-time detection-based tracking models, the most interesting two are presented.

2.5.1 SORT

"Simple Online and Realtime Tracking" [11] is a paper released by A. Bewley in 2016 about a pragmatic approach to the MOT problem in online, real-time detection. Using tools such as Kalman Filter and Hungarian algorithm, the system achieved an accuracy comparable to state-of-the-art online trackers, while being over twenty times faster compared to the other trackers. The key of its high performance, according to the authors, is its simplicity. The model working principle is split into four main stages:

1. **Detection**, being a detection-based tracking algorithm, it is required that the model must work side-by-side with a detector, to feed the tracker with the required data about objects;

2. **Estimation Model**, i.e. the model representing the motion, used to propagate the target's identity into the next frame. The model approximates the inter-frame displacements of each objects with a linear constant velocity model, independent of other objects and from camera motions. The state of each object is modelled as:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \tag{2.14}$$

u and v are representing the horizontal and vertical locations of the bounding box centre, while scale s and r are representing the area and the aspect ratio of the target bounding box. When a detection is associated to a target, the detected bounding box is used to update the target state, where velocities are solved through a Kalman filter framework;

3. **Data Association**, where detections are assigned to existing targets. Each target's bounding box geometry is estimated by predicting its new location in the current frame. The assignment cost matrix that matches detections with existing tracks is computed by the intersection-over-union (IoU) distance between each detection and all predicted bounding boxes from the existing targets. This task is solved optimally by using the Hungarian algorithm. A minimum IoU is imposed to reject assignments where the detection to target overlap is less than IoU_{min};

4. **Creation and Deletion of Tracks**, every track is identified with an id. For every object entering or leaving the scene, a track is destroyed or created. An untracked object is detected when an overlap less than IOU_{min} is detected. The tracker is then initialised using the geometry of the bounding box with the velocity set to zero, initialising also the co-variance matrix with large values, reflecting the uncertainty behind the value. The new tracker undergoes a period where the target needs to be associated with detections to accumulate enough evidence in order to prevent tracking of false positives. Moreover, tracks are terminated if are not detected for T_{lost} frames.

As will be discussed later, both for its simplicity and for the assumptions made, the model is a good candidate for accomplish the task required in the present work.

2.5.2 DeepSORT

Wojke et al. expanded SORT in 2018 with DeepSORT [14], integrating appearance information into the algorithm by means of a CNN for feature extraction, which was trained on a pedestrian detection database in the paper "Simple Online and Realtime Tracking with a Deep Association Metric". The aim of the algorithm is reduce the tracking errors due to occlusions, where SORT was particularly weak since employed association metric is only accurate when state estimation uncertainty is low. The main improvements of the system compared to SORT, can be described into three main points:

1. **Track Handling and State Estimation**, is performed much like is done in SORT algorithm;

2. **Assignment Problem**, done through an extension of the Hungarian algorithm, able to integrate motion and appearance information through combination of two appropriate metrics. While the motion metric is

integrated using Mahalanobis distance, the appearance vector is a vector able to describe all the significant features of a given object. This step is done by employing a CNN-based feature extractor. On the one hand, the Mahalanobis distance provides information about possible object locations based on motion that are particularly useful for short-term predictions. On the other hand, the cosine distance considers appearance information that are particularly useful to recover identities after long term occlusions, when motion is less discriminative. To build the association problem, both metrics are combined using a weighted sum;

3. **Matching Cascade**, the measurement-to-track associations are solved in a cascade fashion that solves a series of sub-problems, giving priority to more frequently seen objects to encode probability spread in the association likelihood. The association algorithm works on a subset of tracks that have not been associated with a detection in the last frames, solving a linear assignment with those tracks and the unmatched detections. The matching cascade gives priority to tracks seen more recently, while in the final stage, intersection over union is run, as seen in SORT, between the set of unconfirmed stages and unmatched tracks of age equals to 1.

The "deep" part of the model is employed in the appearance feature vector, where a offline-trained feature extractor is employed. The presented model was able to track objects for longer periods of occlusions, reducing identity switches of approximately 45%. Despite the implementation of a CNN, the model can still be run in real time.

2.6 Evaluation criteria and definitions

Detection and tracking problems usually employ a specific set of metrics in order for the model to be evaluated correctly. The present chapter defines each metric and presents some general concepts about the object detection and tracking problems.

2.6.1 Intersection Over Union (IoU)

Intersection Over Union (IoU) measures how accurate is the predicted bounding box with respect to the ground truth bounding box. It is defined as the overlapping area between the predicted bounding box and the ground truth bounding box, divided by the area of union between the two. A total overlap outputs an IoU equal to 1 while a null overlap gives 0 as result. Usually a threshold is defined, as a level that considers a detection valid.

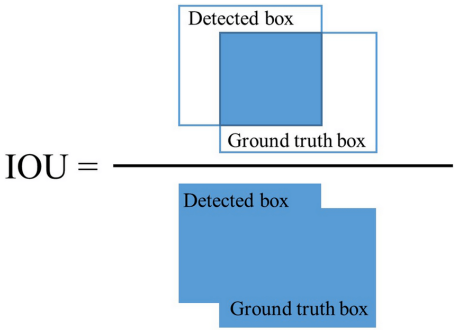


Figure 2.22. Graphical representation of Intersection over Union (IoU)

2.6.2 Precision and Recall

To evaluate the performance and accuracy of the counting system, the following definitions are used:

- **True Positive (TP)**, a correct count;
- **False Positive (FP)**, a wrong count, such as an identity switch;
- **True Negative (TN)**, does not apply in object detection, since it indicates the bounding boxes that should not be detected, basically every other object that is not interesting for the detection process. It is usually neglected;

- **False Negative (FN)**, the amount of missed (undetected) objects.

While *precision* measures how accurate is the count taking into consideration identity switches and double counts, *recall* measures how many objects have been missed during the counting process. The definitions are the following:

$$precision = \frac{TP}{TP + FP} \tag{2.15}$$

$$recall = \frac{TP}{TP + FN} \tag{2.16}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.17}$$

2.6.3 Mean Average Precision (mAP)

Mean Average Precision mAP is the metric used to measure an object detector overall accuracy. The Average Precision (AP) is defined as the area under the precision-recall curve for each label class. The Mean Average Precision mAP is the average of the AP for every class, hence for a detector aimed to detect only one class, AP is equal to mAP. The area under the precision-recall curve is usually approximated through the interpolation of the curves, as can be easily seen in the figure 2.23.

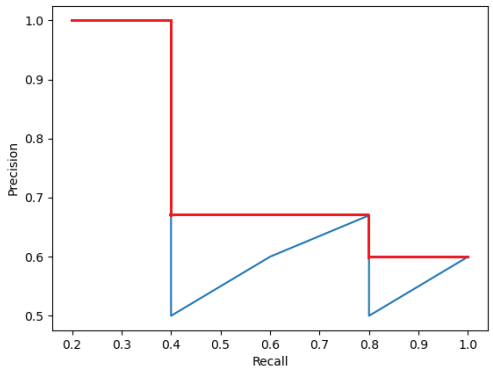


Figure 2.23. Example of precision-recall curve interpolation, to obtain mAP

2.7 Software Platform

The software used for the project development and testing, are represented by the following:

- **Python**, is the selected coding framework for the implementation of the YOLOv4 model and the SORT tracker. Also all the post-process has been done through Python. The most used library have been *Tensorflow*, for the YOLOv4 neural network implementation, and *OpenCV*, for image and video manipulation;
- **Blender**, used to build a photo-realistic virtual environment and simulation system. The latter is a powerful rendering tool that allows advanced rendering and animations, for different types of purposes.

Chapter 3

Implemented solution

This chapter aims to give a general panoramic over the implemented solution, providing with motivation and context for the work. As stated before, the proposed solution sees the implementation of a YOLOv4 object detector, paired with a SORT tracking algorithm. To validate and tune such setup, a set of operations have been done. First, a custom simulation environment has been built in Blender, to run different detections in different lighting conditions. Secondly, the system has been evaluated over a real dataset, counting for ground truth by hand. The last validation step sees the retraining of the YOLO network with a real world dataset built by PIC4SeR, featuring apple orchards near Cuneo [20].
In detail, the development of the system followed the following flow:

1. Creation of the simulated world trough Blender;

2. Integration of the YOLOv4 detector with the SORT tracker, in Python trough Tensorflow implementation;

3. Test and tuning of the system characteristic parameters employing the simulated environment;

4. Validation of the results both in the simulated environment and in real world videos;

5. Performance test after retraining with the custom dataset;

3.1 Detection - YOLOv4

The YOLOv4 network, already discussed previously in the chapter 2, has been chosen since representing the state-of-the-art for object detection in terms of speed and accuracy. The latter has been implemented starting from an open source repository [30], expanded to ensure object detection in videos. The repository is based on a Tensorflow implementation of the network, thus it does not rely directly on the *darknet* repository [32].

The YOLOv4 network features the following tuning parameters:

- $Size_{YOLO}$, the dimension of the image input to be fed into the network;
- $Score_{YOLO}$, the confidence value threshold from where a detection is considered to be valid;
- IoU_{YOLO}, working as a threshold in situations where two overlapping detected bounding boxes must be considered as one;

3.2 Tracking - SORT

SORT has been implemented as tracking framework for the system, since being the state of the art for speed and accuracy for non-deep trackers. The latter in fact were not indicated for the task, since the deep part would have worked sub optimally due to the low variance between apple's features, instead introducing major problems in identity switches and resulting in a slower inference speed compared to SORT. The implemented SORT tracker has been proved to produce satisfactory results even in overlapping object where the latter were characterised by being small and feature similar visual features.

The variables made available for the SORT tracker are:

- IoU_{SORT}, minimum score for IOU match for the predicted bounding box;
- Max_{SORT}, maximum number of frames to keep alive a track without associated detections;
- Min_{SORT}, minimum number of associated detections before track is initialised.

3.3 Counting

Apple counting has been performed exploiting SORT tracker policy for validating a track, performing an unique id counting. In detail, the following procedure is employed:

- A track is considered valid when the IoU of the detection and the predicted bounding box is greater than IoU_{SORT} and when it has being hit by at least Min_{SORT} detection hits. The track is not initialised if not validated by a detection for Max_{SORT} frames;
- For each track, an unique id is defined. For every validated track, its id is recorded on a Python set.
- For each frame, the algorithm counts how many tracks have been validated by SORT on the current frame;
- For each frame, the algorithm counts how many unique ids have been counted up to that frame, thus updating the final count;

The counting algorithm, along with a proper YOLO and SORT parameters tuning, have been proven to work efficiently in those respects.

3.4 Retraining

To further validate the system, the YOLO detector has been retrained with a custom dataset built internally by PIC4SeR, featuring real world apples. The dataset consists of 617 photos and is part of the thesis work made by Debora Cravero done in 2019 [20]. The test dataset features 617 pictures taken from an orchard outside Cuneo, took using the Canon EOS 60D as an acquiring tool in 5184x3456 resolution, at different heights from the ground and different distances from the tree trunk. Various types of apples have been found: Gala, Crimson Crisp, Golden Delicious, Fuji Raku-Raku and Red Chief. Some sample images can are shown in figures 3.1.

All the images have been accurately labeled and classified. Moreover, the dataset has been further improved by the following operations:

- **Resize**, the images have been resized to 832x832, which is the dimension set for the YOLO neural network. By resizing the images previously, the learning algorithm is able to run faster, without any loss of information since the images would have been resized regardless;

- **Data augmentation**, in particular the operations done on the dataset have been "Flip" (image mirroring), "90° rotate" (image rotation), "Crop" (image zoomed crop) and "Cutout" (random black spots in the image). The operations affect the image in its full dimension.

The dataset at this point contains over 1543 images. The tool used in order to perform such changes is Roboflow [33], a dataset manager and editor service available on internet. The retraining has been done through the use of *darknet*, using a virtual hardware made available trough Google Colab [29].

Figure 3.1. Examples images taken from the training dataset

3.5 Test set

To test and tune the developed system, two set of validations has been employed:

- **Simulation environment**. The simulated world described in 4 has been employed to run a total of 6 simulation batches, equal to three video batches for green apples and three video batches for red apples. While the positions of the apples have been randomised between the batches, each batch is composed by eight different simulations, corresponding of eight different day times. Thus in total, 48 videos have been produced and evaluated for the test. The total amount of apples for ground truth is fixed to 583 apples. This controlled environment is useful to obtain ground truth information to evaluate the detector and tracking performance, evaluate the precision of output count figure produced by the counting algorithm and evaluate for the output variance when varying working conditions such as lighting and apple positions. All the videos have resolution of 1280x720 and are recorded in 30fps;

- **Real world environment**. The results obtained in the simulated world are compared to the results obtained in three real-world videos, in order to assess the system performance on real world conditions and evaluate the system robustness. Three videos real-world videos for every environment have been employed, for a total of nine videos. The videos have been recorded with a optically stabilised GoPro 9 Action Camera in 2160x3840 resolution, scaled down to 1280x720, in three different environments, camera pose, locations and light conditions. The ground truth has been derived counting by hand every apple object in the video. To minimise human errors, the real world videos forty seconds videos have been split into segments lasting five seconds. For each segment, the count has been carried out three times for each segment and the total apple count have been derived summing up each segment contribution, thus obtaining the final number of apples counted in the whole video. No ground truth bounding box information is derived for the real world videos for time and operational difficulties that would have made this practically impossible;

Figure 3.2. Example of the dataset images after data augmentation

Chapter 4

Simulation

This chapter is dedicated at detailing the simulation system developed in order to properly validate the detection and counting infrastructure. The virtual environment has been built to resemble a real world situation, also accounting for surface roughness, fruit location randomisation and eight different lighting conditions. The instrument used to perform such a task is Blender, which represent one of the best tools for computer rendering.

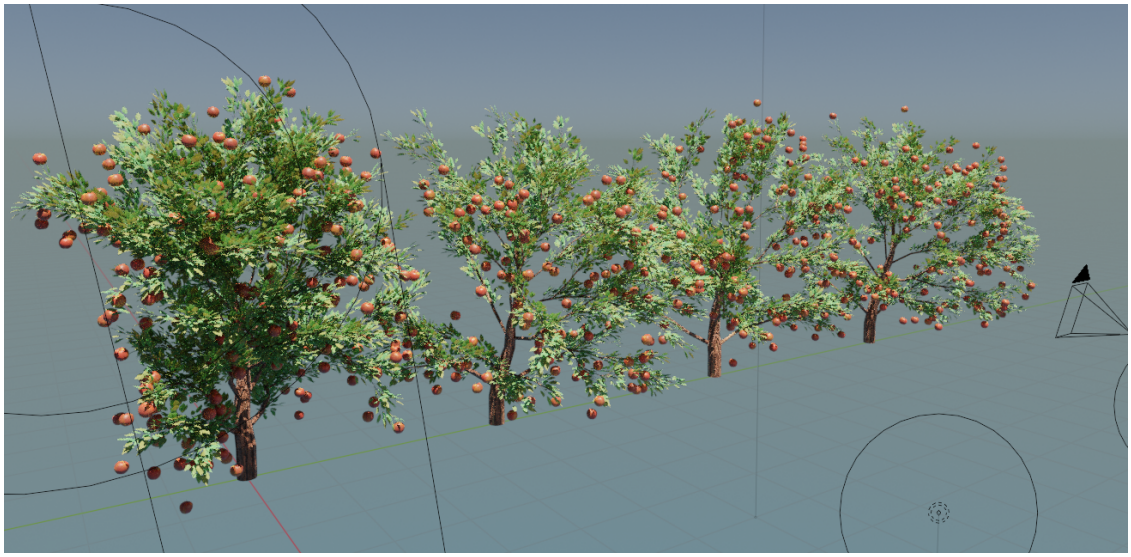


Figure 4.1. The virtual environment built in Blender

4.1 Objectives and outputs

A custom virtual environment has been needed in order to correctly evaluate the results given by the system, being the ground truth the most important parameter to correctly evaluate the counting precision. For the real world videos the ground truth must be derived by hand through manual counting, obviously introducing major errors in the ground truth count while also featuring minor performance reproducibility. The simulated environment not only is useful to check the accuracy of the final apple count based on the number, but is also useful to account for double counts, false positives and missed detections thanks to the availability of the ground truth bounding boxes. Thus, the simulated virtual environment has been developed to achieve the following objectives:

- Obtain a realistic video reference that can work as test to test, tweak and tune the tracking and counting system;
- Account for different apple types, such as green and red ones;
- Account for different type of lighting conditions in order to test the system robustness;
- Account for real world non-idealities such as apple pose randomisation and surface roughness;
- Obtain ground truth total count values and bounding boxes info, in order to assess the system precision, recall and identity switches figures;

As for the outputs, the simulated virtual environment must output the following:

- A set of eight different 40 seconds long videos, representing the rendering of the virtual orchard in different lighting conditions;
- The ground truth total fruit count;
- The ground truth fruit detected in frame;
- The ground truth bounding boxes;

4.2 Features

Every simulation batch consists of eight different videos, that depicts the same apple orchard with fruits in the same position, simulated in eight different day time hours. Also, every batch runs both for red apples and for green apples, for validations purposes. The shade of the two apple types have been accurately chosen from real world benchmark images. The virtual environment built must achieve the objectives detailed in the present section, where every feature is taken into consideration. The following details can also be taken as a reference for the real world implementation of the system and for future developments of the present work.

4.2.1 Structure and location

The virtual apple orchard is structured in order to resemble a real world one. For such task, a reference orchard has been taken into consideration. In particular, the tree structure used is the *Malus Pumila*, which is one of the more common on the open field cultivation of apples. In order to give a reference for the reader and in order to reproduce the results shown, figure 4.2 represents accurately the geometrical disposition of the objects in the scene. Taking into consideration the most common structure seen on real apple orchards, the simulated environment features 4 trees distributed over a 12 meters long strip of crop, as seen along the blue axis in figure 4.2. The strip features only four trees since to reduce rendering time while still have a realistic environment. The camera, virtually mounted on a robot, travels along the strip at a speed of 0.4 m/s, starting 2.5 meters before the first tree and 2.5 meters after the last one, thus having a total travelled distance of 17 meters, represented by the red axis in figure 4.2. In the orchard, each tree is placed 4 meters away a part from the other. The trees are 5 meters high, while the apple is about 7cm of diameter.

4.2.2 Camera positioning

The camera position has been accurately studied in order to find the best solution to maximise the apple dimensions on the camera image, trying to have every apple in the camera viewport. In fact, as explained over in the final part of the work, the situation where apples object are small is the one where the system performs the worst. Moreover, since the average apple orchard is quite tall, the camera is rotated in its vertical pose in order to

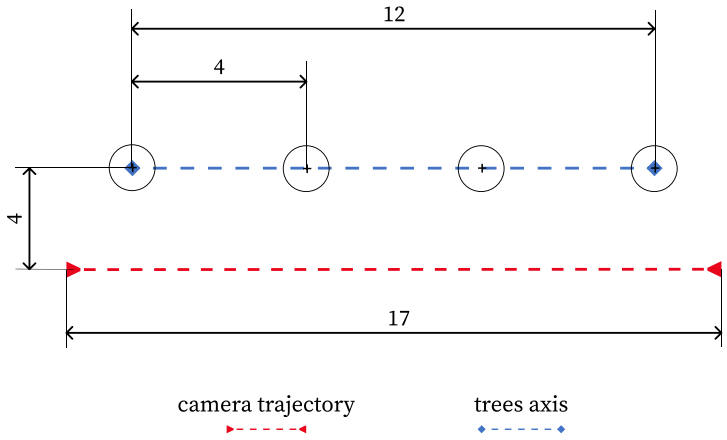


Figure 4.2. Top view of the simulated environment

maximise the amount of apples the system can cover for every frame. In terms of geometry, the camera has been placed 4 meters far away from the apple tree at an height of 1.5 meters, with a vertical rotation of 14° degrees, as seen in the picture 4.3. Moreover, the lens field of view (FOV) has been increased to a focal lenght of 37mm, simulating the so-called "fish-eye" that is common in the world of portable or action cameras and allows for a wider camera.

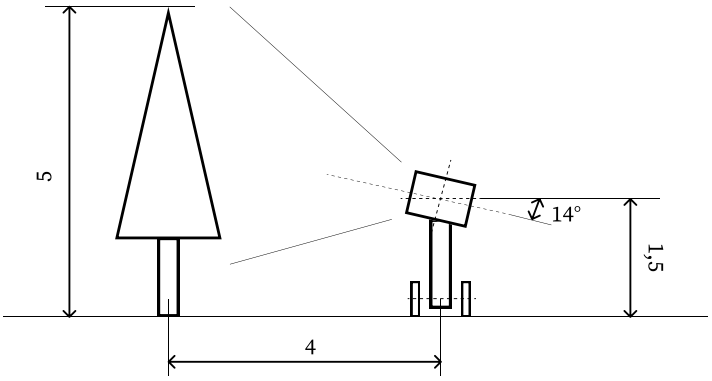


Figure 4.3. Side view of the simulated environment

4.2.3 Light conditions

In order to validate and test the system in various light conditions, the virtual environment has been developed so that it simulates eight different light conditions, equivalent to eight different day hours. In particular, the selected hours range starts from 6 AM and goes through 6 PM with an interval of 2 hours, with a bonus scenario that resembles midnight, with a lamp mounted on the camera. A sample picture of every scenario can be seen in the figure 4.4.

Figure 4.4. The virtual environment depicted in the various lighting conditions

For every selected hour, the following settings change automatically:

- Sun absolute position on the virtual world, resembling its true position for sunrise, from mid-day to sunset;
- Sun rays rotation angle, resembling its true angle with respect to the sun position;
- Sun light power;
- Sun light colour, to account for the colour changes due to sunset or sunrise;
- Sky colour, to simulate the darker day times such as sunset or sunrise;

The sun has been implemented in the environment using the custom *Sun* object in Blender, located in the world as seen in the figure 4.5. The location of the sun mimics the sun dynamics, reasonably approximating real-world horizon elevations degrees for Rome (IT) in the day of summer solstice, June 21st.

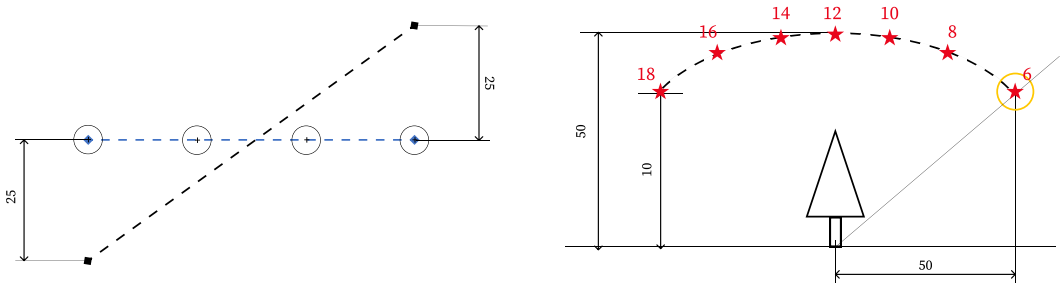


Figure 4.5. Top view and side view of the sun dynamics implementated in the simulated world

4.2.4 Surface roughness

To simulate the roughness and disconnections of the surface, a two step approach has been used. Firstly, the roughness has been accounted using the ISO 8608:2016 norm [10], that regulates how surface roughness is addressed for road modelling. This first layer accounts for the macro-deviations on the road profile. The norm specifies a uniform method of reporting measured vertical road profile data, based on the vehicle velocity and the Power Spectral Density (PSD) of the road irregularities. Eight different classes (from A to H) are identified in order to classify road profiles based on the roughness and the PSD. The class considered for this work belongs to the E class, the one featuring the highest displacements for road irregularities.

The second step is used to reproduce surface graininess due to rocks or smaller holes in the ground, also providing for an additional step of disturbance. The second layer is represented by a randomly-generated white noise signal superimposed to the previously generated signal.

4.2.5 Location randomisation

In order to validate the efficiency of the model, every eight simulation of a single batch features the same apple position. Between different batches

however, the system generates a random apple position for every apple object by offsetting a standard "default" position of a random number. The randomisation also affects the apple rotations. The numbers generated have been carefully tuned in order to result in realistic apple positions.

4.2.6 Output info

Every simulation batch, outputs the following data for every of the eight videos produced:

- A file containing the total apple count up to the considered frame and the apple count in the considered frame, for every frame of the video;
- A file containing every ground truth bounding box position and the ground truth tracking number of the corresponding apple, for double counting assessment purposes;

Every file is contained into a comma separated value (.csv) file, produced in the folder were the rendered videos are stored.

4.3 Blender setup

To have the system detailed above working in the proper manner, a compelling Blender environment was been created with several objectives in mind. As for the graphical part, the world must be as photo realistic as possible, as much for the apples as for the tree part. For such a reason, a specific add-on library called "MTree" [31] was employed in order to generate random trees that resemble the real counterpart.

The add-on library also allows for tree customisation, such as foliage structure and dimension tuning. The apples were positioned by hand at first, in order to make sure to have a compelling positioning with respect to real world cases. The amount of apples for every tree has been evaluated taking into account the real world counterpart. Once the world is created correctly, the final rendering must be produced. Blender has its own animation engine to render videos, but given the complexity of the environment and all the requirements to be satisfied, a customised rendering script has been developed. The latter renders every frame of the final video and mounts the final video file, looping through every of the eight light conditions, and taking into account every of the features detailed in the previous chapter. The script is 700 lines long and it is divided into various parts:

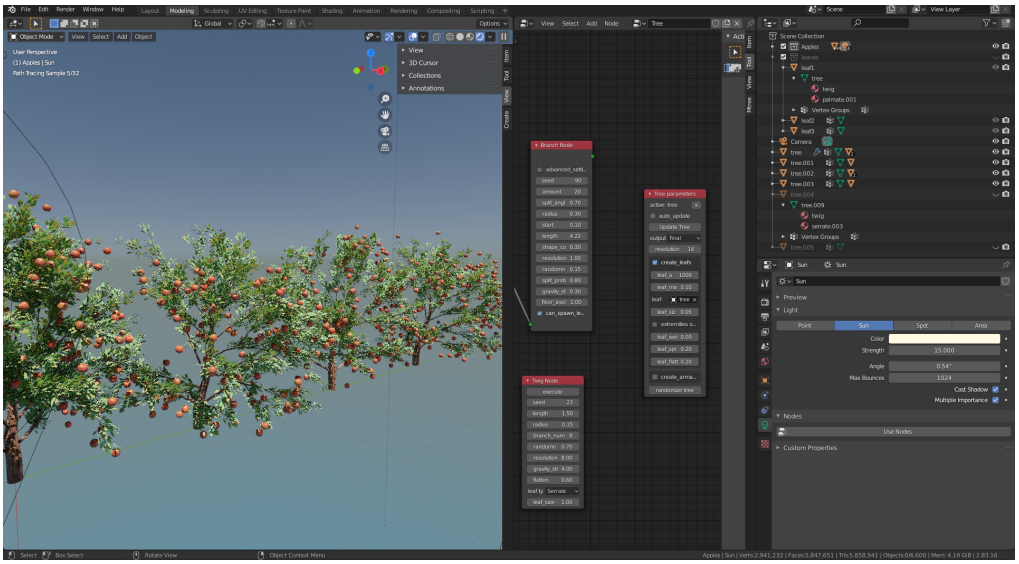


Figure 4.6. The Blender software view of the simulated world

1. Blender settings setup;
2. Folder structure setup;
3. Camera location setup and road profile initialisation;
4. Rendering steps calculation;
5. Apple location randomisation;
6. Sun setup;
7. Rendering;
8. Ground truth detection;
9. Output file update;
10. Output video rendering;

Moreover, on the script, a vast number of lines have been dedicated to properly log the current state of the rendering during the rendering itself, as well as detect potential errors.

Chapter 5

Results

In this chapter, the development process of the system is detailed, commenting every design choice made and showing the results obtained with the system. The development flow of the system, as stated before, first started with some test having the objective of testing the system performance and tune its main parameters, in order to find a setup that achieves satisfactory performance. The test have been carried out on the simulated environment. After that, the performances have been evaluated on the real videos. The final stage accounts for performance differences after the network retraining.

The hardware used for the following performance evaluations features a Quad-core Intel® Core™ i7-6500 @ 3 GHz, 16.0 GB RAM and NVIDIA GeForce GTX 1070, while the input videos are in 720p format, in 30FPS. Moreover, as detailed in the chapter 4, every result shown in the following section is the average of one so-called *simulation batch*, that is composed by the eight different videos in the eight lighting conditions.

5.1 Parameters tuning

The goal of the initial phase in the development of the present work, is to find the correct network setup that is able to perform the wanted task with satisfactory results in terms of precision and speed. The set of parameters to be tuned are the following:

1. $Size_{YOLO}$, the dimension of the image input to be fed into the network;
2. $Score_{YOLO}$, the confidence value threshold from where a detection is considered to be valid;

3. IoU_{YOLO} in situations where two overlapping detected bounding boxes must be considered as one;

4. IoU_{SORT}, minimum score for IOU match for the predicted bounding box;

5. Max_{SORT}, maximum number of frames to keep alive a track without associated detections;

6. Min_{SORT}, minimum number of associated detections before track is initialised.

While some parameters are considered to be almost standard, such as IoU_{YOLO}, IoU_{SORT} and Max_{SORT}, the others are affecting significantly the output performance. The following section is devoted to detail the test process and show how the system perform based on various parameters.

5.1.1 Model size

The input size of the network is one of the most influential parameters both in terms of precision and in terms of inference speed. A system with an high input dimensional is able to detect smaller objects and achieve overall better results, while however featuring an higher inference time for the detection task. To evaluate both precision, recall and inference speed, the system has been tested with the following input sizes: 416x416, 512x512, 608x608 and 832x832, both for YOLOv4 and its simplified version, YOLOv4-tiny. While the tests have been carried over a simulated video, the IoU_{YOLO}, IoU_{SORT} and Max_{SORT} parameters for testing have been set according to the table 5.1.

System parameters	
Parameter	Value
$Score_{YOLO}$	0.2
IoU_{YOLO}	0.4
IoU_{SORT}	0.3
Max_{SORT}	70
Min_{SORT}	10

Table 5.1. System settings for YOLO confidence parameter tuning

The results can be seen in the figures 5.1 for YOLOv4 and 5.2 for YOLOv4-tiny.

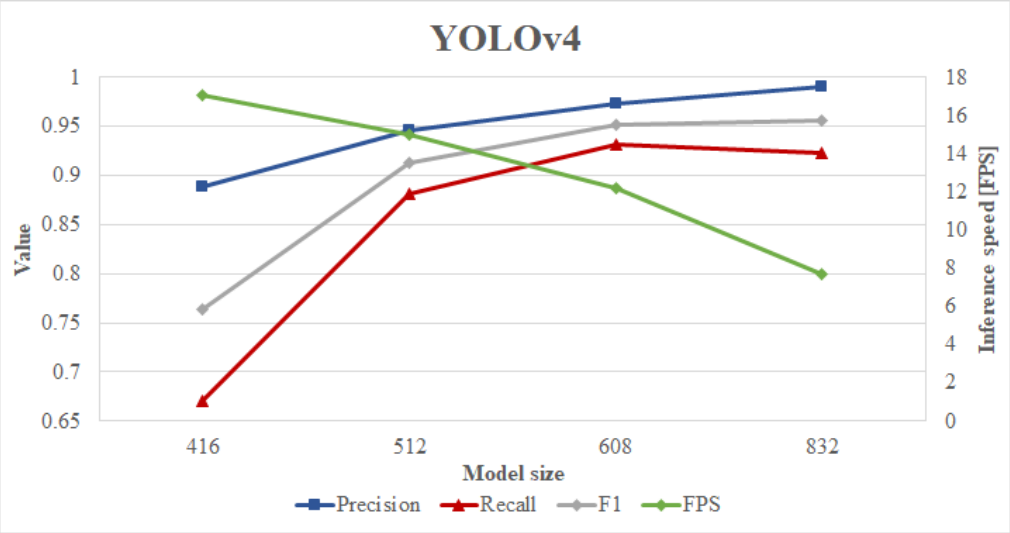


Figure 5.1. YOLOv4 model size performance and speed

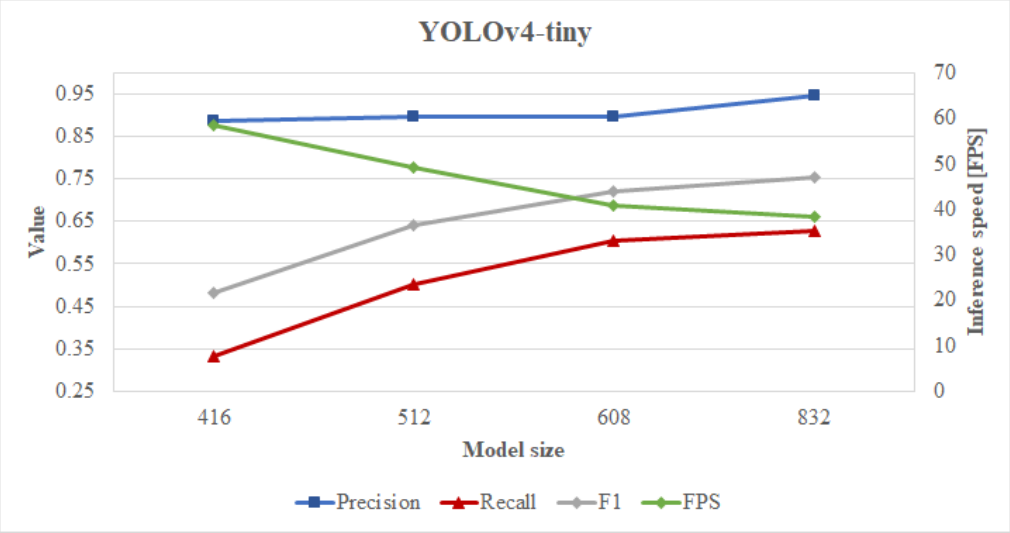


Figure 5.2. YOLOv4-tiny model size performance and speed

It can clearly be seen that, while YOLOv4-tiny is able to feature inference speed up to 60FPS, it does not provide with satisfactory F1 and recall figures, the least being 0.55. The goal of the present work is to develop a much more

accurate system, thus YOLOv4 has been considered. The 608 model size is considered the best since 832 employs minor performance increases at the cost of a much higher computational cost, paid in terms of inference speed. It must be noted also that recall is slightly lower in the 832 model, to be attributed to identity switches during the counting process.

5.1.2 Detection and tracking parameters

The other parameters that highly affects the system performance are $Score_{YOLO}$ and Min_{SORT}. Max_{SORT} plays a less important role on the tuning of the system. In order to carry out the tests described in the following paragraphs, the system featured the parameters depicted in table 5.2.

System parameters	
Parameter	Value
$Size_{YOLO}$	608x608
IoU_{YOLO}	0.4
IoU_{SORT}	0.3
Max_{SORT}	70

Table 5.2. System settings for YOLO detection and SORT tracking parameter tuning

YOLO confidence score

The confidence score of the YOLO detector is one of the most important parameter of the system, which is also placed on top of the parameter chain, meaning that this parameter influences on cascade every other in the system.

$Score_{YOLO}$, is the confidence value threshold considered from YOLO network that allows the detector to consider the detection as valid. The YOLO system outputs a prediction in the form of a bounding box and a confidence representing the probability from 0 to 1 that that the object is correctly described by the detection in terms of bounding box and label.

Thus, the parameter has been compared to the precision and recall in order to correctly evaluate the right choice for the system, resulting in the overall result seen in figure 5.3.

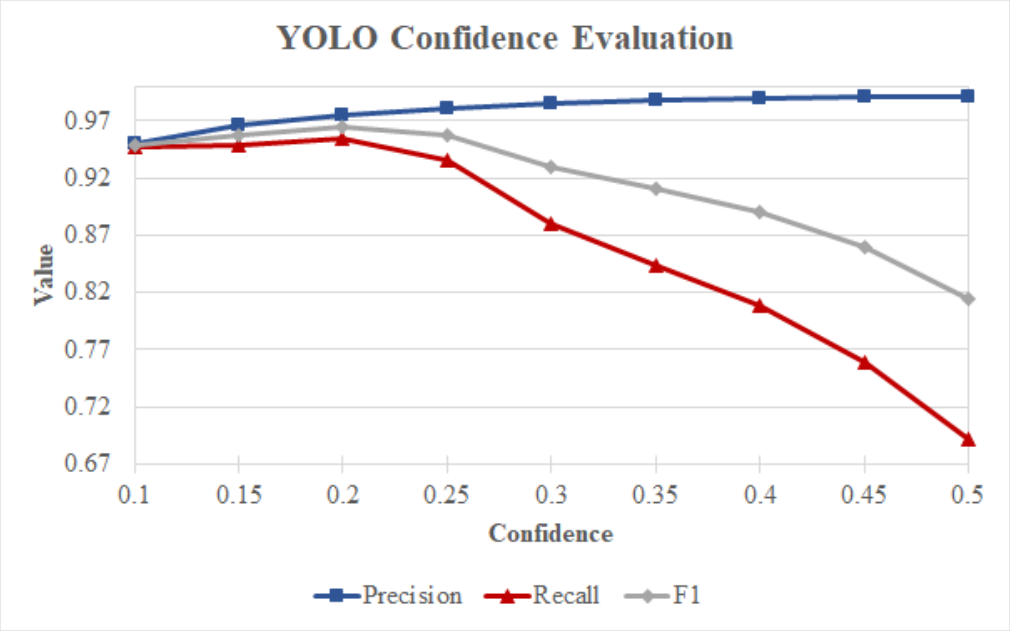


Figure 5.3. YOLO Confidence evaluation

The trend shows that over a $Score_{YOLO} = 0.2$, the precision of the system increases while the recall starts gradually decreasing. Since the recall figure is related to the amount of missed apple counts, the result shows an expected behaviour. Over a certain threshold, the detector recognises as valid objects less and less fruits, thus resulting in higher missing counts. However, the objects detected with higher confidence, are the ones who show an higher precision figure, related to how many of the detections are actually correct.

Since the objective of the system is to minimise the amount of missed objects, thus to maximise the recall, the figure 5.3 highlights that the best parameter for achieve this task is $Score_{YOLO} = 0.2$.

SORT minimum hits

Min_{SORT} is a SORT parameter that represent the minimum number of associated track detections before the given track is initialised. An high value of the parameter means that the track requires more validations to be initialised, thus lowering identity switches, identified in the False Positives (FP) counts. A lower value implies fewer missed counts, identified in the False Negatives (FN) counts, since the object takes less frames to be validated. The figure 5.4 highlights the results obtained.

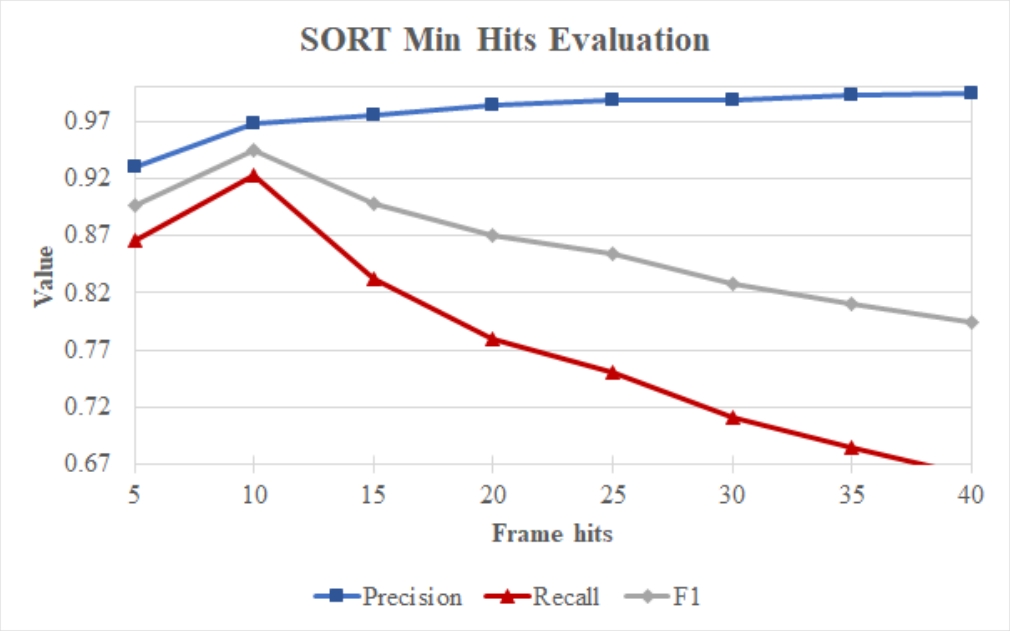


Figure 5.4. SORT Min Hits evaluation

As seen before, an optimal value is found in $Min_{SORT} = 10$, being the threshold where False Positives and False Negatives are minimised, thus outputting the best recall and precision figures.

SORT maximum history

The last parameter to be tuned is Max_{SORT}, that is the number of frames a track must be kept alive even without any associated detection. This figure affects False Positives (FP) counts since for extremely low values, a track can be discarded and subsequently re-initialised causing a double count. An high value implies that every track is taken into considerations for a longer number of frames. This parameter has been proven very effective on dealing with occlusions. As expected, no significant difference on the system is found after raising the parameter after a certain value. The chosen value for the parameter is thus $Max_{SORT} = 70$

5.1.3 Final setup

After all the above testing being done, the system has successfully been proven to work efficiently with the setup in the table 5.3, that have being used for the whole thesis work.

System parameters	
Parameter	Value
$Size_{YOLO}$	608x608
$Score_{YOLO}$	0.2
IoU_{YOLO}	0.4
IoU_{SORT}	0.3
Max_{SORT}	70
Min_{SORT}	10

Table 5.3. Final system parameter setup

5.2 Standard YOLOv4

5.2.1 Simulations

In order to validate both the algorithm and the system, a total of 6 *simulation batches* have been performed, equal to three video batches for green apples and three video batches for red apples. The positions of the apples have been randomised, as stated in the chapter 3. Each batch is composed by eight different simulations, corresponding of eight different day times. Thus in total, 48 videos have been produced and evaluated. The present results have been derived with the standard pre-trained YOLOv4 network, trained on MS COCO class set. The ground truth total apple count is fixed to 583 apples, for every video of the simulation. Thus, the results of each batch can be summarised in the table 5.4.

Averaged results values								
Hour	Count	Error	TP	FP	FN	P	R	F1
6	570	2.32%	558	12	25	0.979	0.957	0.968
8	570	2.32%	555	15	29	0.973	0.951	0.962
10	552	5.40%	542	10	42	0.982	0.929	0.954
12	555	4.89%	549	6	35	0.989	0.941	0.964
14	542	7.12%	535	7	49	0.987	0.917	0.951
16	539	7.63%	529	10	57	0.982	0.907	0.943
18	537	7.89%	527	11	57	0.981	0.903	0.940
24	534	8.49%	518	16	66	0.970	0.888	0.927
avg	**549.4**	**5.76%**	**538.7**	**10.7**	**44.2**	**0.981**	**0.924**	**0.951**

Table 5.4. Standard YOLOv4 results on simulations

The column *Hour* refers to the defined day hour, the *Count* column is representing the output number given by the counting algorithm, with the relative error with respect to the ground truth indicated in the column *Error*. False Positives, *FP*, are representing the identity switches counts, number of apples mistakenly counted. False Negatives, *FN*, are representing missed apples, not counted by the system. Analysing the obtained metrics, the following considerations can be made:

- The results show state-of-the-art metrics in terms of precision and recall, resulting in a representing a system that can be used effectively for apple counting tasks;

- The results show consistent metrics over the various light conditions, ranging with F1 results ranging from $F1_{max} = 0.968$ for the best case and $F1_{min} = 0.927$ for the worst case. In particular, after mid-day, the darker ambient light penalises the counting process, with higher missed counts and higher false positives. The mid-night environment, even with a lamp mounted on the rover, does not seem to provide better results compared to daylight conditions. The precision and recall errors however seem to be consistent with the environment conditions. The figure 5.5 highlights much smoother and coherent performance figures with respect to daytime hours, still pointing out that day-light allows for the best performance of the system and mid-day is the best working condition;

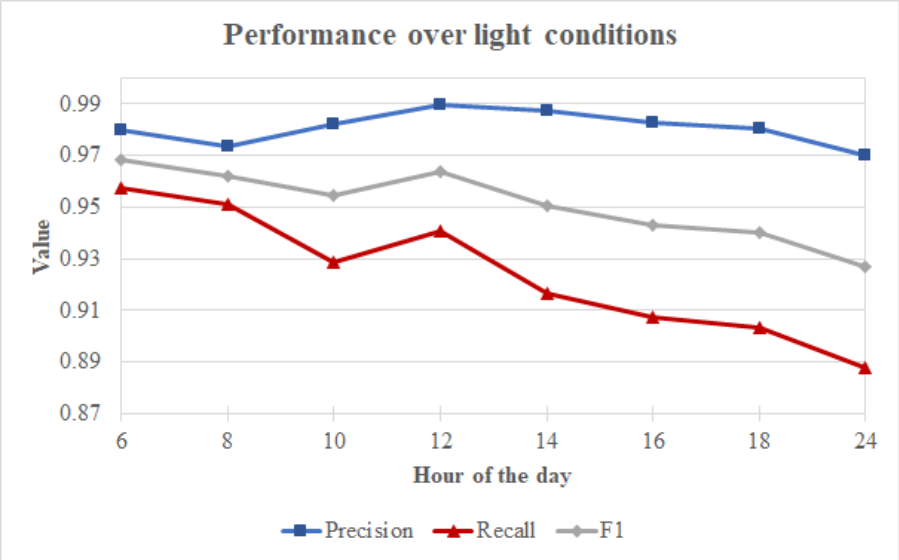


Figure 5.5. Performance metrics of the standard YOLO network system as the hour of the day

- The count figure, seen in the *Count* column, produced by the system is reasonably accurate, since on average the system detects only 10.7 double counts or identity switches, for over 550 total apple objects counted. Its accuracy can be assessed looking at the precision number, that measures how many of the counts are valid (True Positive) over the total counts performed (True Positives plus False Positives), as seen in chapter 2.6.2. The recall number is reaching 0.981 on average, meaning only 2% of the counts are considered as wrong, which is considered as a satisfactory

result. False positives are due typically to object or trajectory overlap. Also in the case of long occlusions, in terms of tens of seconds, are outputting identity switches. False Negatives must be attributed to major leaf occlusion or completely covered apples.

The inference speed for the system has recorded to be on average 11.54 FPS. The figure 5.6 represent two frame captures of two videos during the detection phase, one for red apples and one for green apples.

Figure 5.6. Snapshots taken from the simulated environment during the detection

5.2.2 Real videos

In order to assess the system performance on real world conditions, three real-world videos have been employed. The videos have been recorded with a optically stabilised GoPro 9 Action Camera in 2160x3840 resolution, scaled down to 1280x720, in three different environments, camera pose, locations and light conditions. In order to properly assess the system, a way to derive the ground truth must be derived. In this case, the fruit objects in the video have been counted by hand. To minimise human errors:

- The real world videos forty seconds videos have been split into segments lasting five seconds;
- The count has been carried out three times for each segment;
- The total count of apples have been derived summing up each segment contribution, thus obtaining the final number of apples counted in the 40-second long video.

The information about bounding boxes and thus False Positive and False Negatives have been neglected, due to the high amount of manual data needed to be produced, that would also have faced major issues due to human error. Also in the real world videos a total of three videos for every scene have been used, for a total of nine videos. The results of each batch can be summarised in the table 5.5.

The output data highlights the following keypoints:

- The results show a robust and consistent performance over ground truth reference in the first two environments, with the counting error floating from 7% to 13% in the two conditions. This represent an acceptable result, since the videos have been taken in different lighting conditions but more importantly, not every video has featured the exact same pose used in the simulation field. In fact, the fruit object size in the viewport is affecting widely the output result: small apples represent a problem for the detector in terms of detection consistency and tracking;
- The third environment highlights the latter effect, where in the video the camera is located very far away from the crop, while also being located at a lower height, while also introducing major fish-eye distortion on the image. This produces a video with the vast majority of apple objects that are very small, also hardly recognisable from background because of the dark colour of the apple itself, with the results of a vast portion

Averaged results values			
Segment	Ground truth	Count	Error (%)
Sequence 1A	399	361	9.52%
Sequence 1B	532	476	10.53%
Sequence 1C	560	489	12.68%
batch	**1491**	**1326**	**11.07%**
Sequence 2A	690	598	13.33%
Sequence 2B	592	539	8.95%
Sequence 2C	617	571	7.46%
batch	**1899**	**1708**	**10.06%**
Sequence 3A	590	432	27.39%
Sequence 3B	595	441	25.25%
Sequence 3C	588	407	30.78%
batch	**1773**	**1280**	**27.81%**

Table 5.5. Standard YOLOv4 results on real videos

of the apples in the viewport not recognised, as can be seen in the table 5.5 under the third sequence;

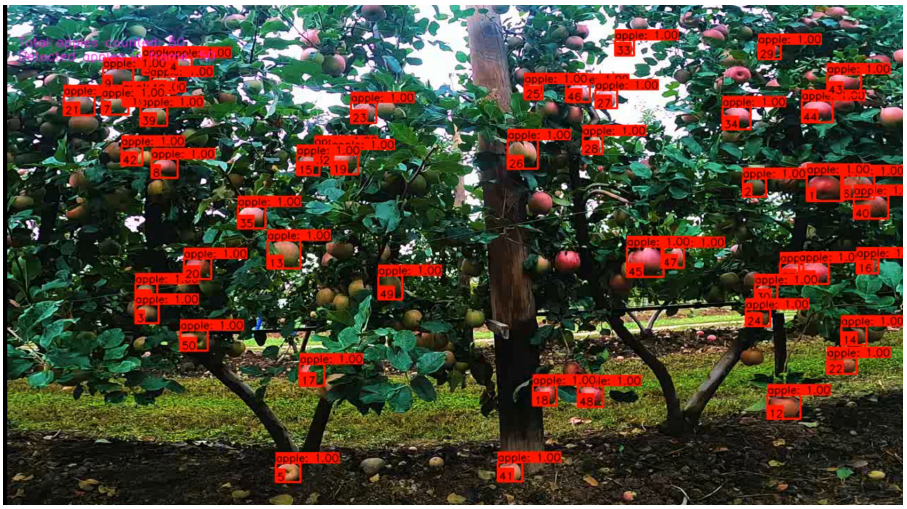


Figure 5.7. Snapshot during detection in the real environment 1

These results can only be taken as indicatives, both because of human error in deriving ground truth, both because some of the detections can be affected by double counts errors. The third environment The inference time for each segment has been on average 10.43 FPS, thus slightly slower than the simulated case. The figures below represent three frame captures of the three environments during the detection phase.

Figure 5.8. Snapshot during detection in the real environment 2

Figure 5.9. Snapshot during detection in the real environment 3

5.2.3 Comparison

While the results shows consistent performance between the simulated environment and the real world videos, a slight worse performance can be denoted on the latter. This is due to the non optimal working conditions, such as lighting and camera pose. While video quality and apple colours can play a role in the worse performance figures, the main responsible has to be attributed to apple size in the viewport, thus the camera position. Bigger objects are handled more efficiently by the system, as can be seen in the comparison from the sequences in the first and second environment, compared to the third. Moreover, the count figure produced in output by the counting system, as seen in the simulation, is a reliable metric of the real number of apples, showing a $\approx 2\%$ error over the performed counts, as seen by the average precision. The latter is dependent on the number of False Positives (FP), the number of detected counts that are not valid due to double counts or identity switches. As for the total counting effectiveness, a $\approx 7.5\%$ recall error is shown, meaning that the system is able to detect on average 92.5% of the fruits present in the orchard. These results are considered as satisfactory and aligned with both the state-of-the-art and the objectives.

5.3 Trained YOLOv4

5.3.1 Training

As stated in the chapter 3, to further validate the system, the detector network has been re-trained over a specific dataset built internally by PIC4SeR, featuring real world apples of various types, such as Gala, Crimson Crisp, Golden Delicious, Fuji Raku-Raku and Red Chief, consisting in a total of 617 pictures. The dataset has been resized to 832x832 and has seen data augmentation implemented, reaching almost 1543 pictures to be evaluated. The dataset has been split in a 75:25 fashion, so that 75% of the dataset is devoted to training while 25% is applied for testing and validation.
Due to the high computational demand required by the process, the training has been carried out using a virtual hardware made available trough Google Colab [], that is a virtual environment able to employ high-end GPU hardware through the web, featuring GPUs such as Nvidia K80s, T4s, P4s and P100s. The retraining process has been done through the use of *darknet*, which is an open source neural network framework written in C and CUDA.
The hyper-parameters of the network have been set up for the training in the following way:

- **Batch size**, defined as number of the dataset's images used for each epoch. The batches have been set to 64, while the subdivisions have been set to 16;

- **Learning rate**, during the training the learning rate changes following the Gradient Descent Theory and the best-in-class techniques proven to produce satisfactory results. The learning rate starts its so-called *warmup* phase, ramping from 0 to $1 \cdot 10^3$ in the first 1000-th iterations. Then starts its updating phase, based on the following rule:

$$L_{n+1} = \begin{cases} d \cdot L_n & n \in [steps] \\ L_n & otherwhise \end{cases} \tag{5.1}$$

  With n representing the iteration index, L the learning rate and d the so-called *decay rate*, which is a parameter between 0 and 1, in this work it has been set to $d = 0.1$, while $[steps] = [4800,5400]$ is a set of two iteration steps that indicates when the learning rate decreases. The curve followed by the parameter is depicted in the figure 5.10.

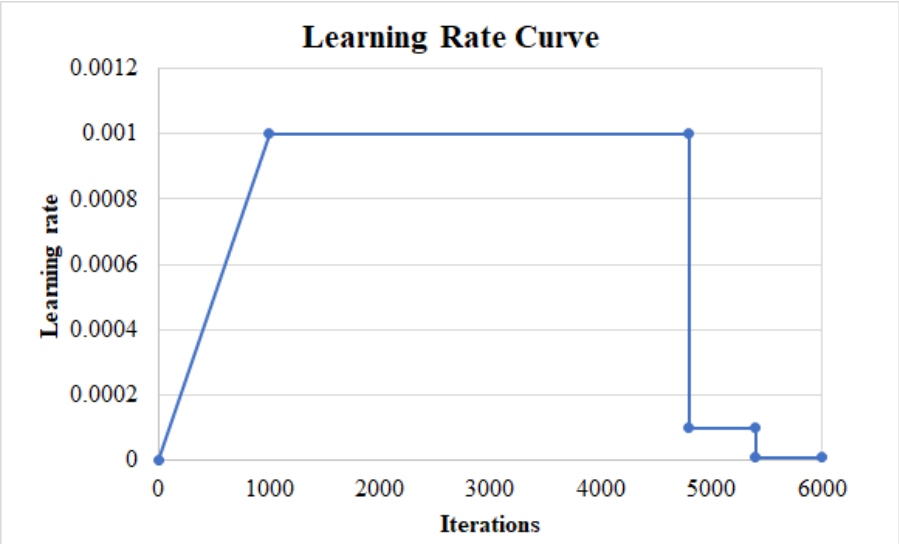


Figure 5.10. Learning rate curve variable along with iterations

The hyper-parameters used for the learning process are thus summed up in the table 5.6. The training process took about 32 hours to complete, with over 6000 iterations that for a 64 batch size, are equal to almost 93 epochs. The loss function behaviour over iterations can also be seen in the figure 5.11, where it can be seen that the training process reached satisfactory results. The final AP precision over the validation set has been 87,64%.

Training hyper-parameters		
Parameter	Symbol	Value
Size	S	608x608
Classes	C	1
Batch size	B	64
Subdivision	B_s	16
Decay rate	d	0.1
Steps	$steps$	[4800,5400]

Table 5.6. Training hyper-parameters

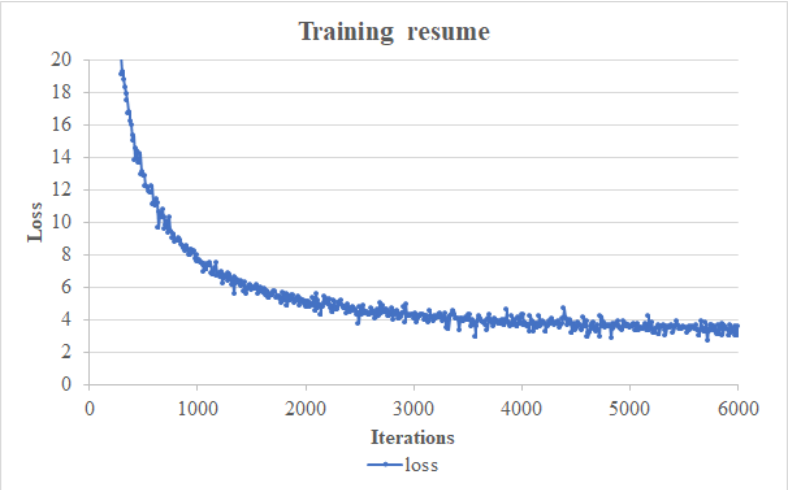


Figure 5.11. Loss curve trend with iterations

5.3.2 Simulations

As seen in chapter 5.2.1, the group of 48 videos featuring the simulated environment have been tested in the same manner, using the re-trained network as a detector, in order to evaluate for performance improvements. Are shown in the table 5.7.

Averaged results values								
Hour	Count	Error	TP	FP	FN	P	R	F1
6	548	6.09%	538	10	46	0.982	0.922	0.951
8	554	5.06%	546	7.5	37	0.986	0.937	0.961
10	562	3.69%	554	6.5	30	0.988	0.952	0.967
12	566	2.92%	561	5.5	23	0.990	0.961	0.975
14	548	6.09%	540	7	43	0.987	0.927	0.956
16	547	6.17%	539	8.5	45	0.984	0.924	0.953
18	541	7.29%	532	9	52	0.983	0.912	0.946
24	540	7.46%	529	10.5	54	0.980	0.907	0.943
avg	**550.3**	**5.60%**	**542.3**	**8.0**	**40.6**	**0.985**	**0.930**	**0.957**

Table 5.7. Trained YOLOv4 results on simulations

The following considerations can be made on the results:

- The results still represent a state-of-the-art systems in terms of precision

and recall, with improvements on both of the metrics. In particular, the number of False Positives have decreased of over 25% on average and False Negatives FN by 8%, with respect to the standard network. Since the new dataset is directly trained with semi-occluded apples, the enhancement was expected both precision and recall figures;

- As seen also in the standard network, the results show consistency over light conditions, favouring day hours from the dark light conditions seen after mid-day. F1 metrics range from $F1_{max} = 0.959$ for the best case and $F1_{min} = 0.924$ for the worst case. Mid-night environment still does not provide better results compared to daylight conditions. The figure 5.12 highlights much smoother and coherent performance figures with respect to daytime hours, pointing out that day-light allows for the best performance of the system and mid-day is the best working condition;

- The improvements compared to the standard network are modest. While still relevant, are not changing substantially the results previously obtained. The improvements sees a 0.41% gain in the precision, from $P_{std} = 0.981$ to $P_{train} = 0.985$, while the recall improves of 0.65%, from $R_{std} = 0.924$ to $R_{train} = 0.930$.

The inference speed for the system has recorded to be on average 12.78 FPS.

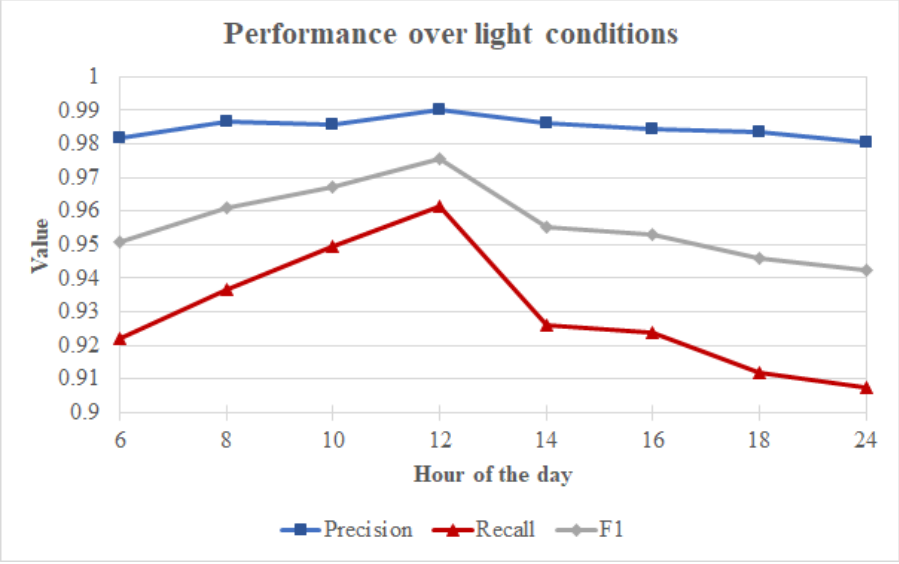


Figure 5.12. Performance metrics of the re-trained YOLO network system as the hour of the day

5.3.3 Real videos

As seen in chapter 5.2.2, the system featuring the trained network has been tested in the three real environments, comparing the performance previously obtained and validating the simulation results. As seen before, the results are shown in the table 5.8.

Averaged results values			
Segment	Ground truth	Count	Error (%)
Sequence 1A	399	363	9.02%
Sequence 1B	532	479	9.96%
Sequence 1C	560	508	9.29%
batch	**1491**	**1350**	**9.46%**
Sequence 2A	690	638	7.54%
Sequence 2B	592	535	9.63%
Sequence 2C	617	571	8.16%
batch	**1899**	**1744**	**8.16%**
Sequence 3A	590	524	11.93%
Sequence 3B	595	500	15.25%
Sequence 3C	588	499	15.14%
batch	**1773**	**1523**	**14.10%**

Table 5.8. Trained YOLOv4 results on real videos

The major keypoints of the resulting data are:

- The results of the network in the real environment show consistency over the simulated world, with comparable performance metrics. The counting error is floating from 5% to 13% in the two videos, with up to 15% performance improvement with respect to previous counting precision. The higher error with respect to the simulation data is relative still to different lighting conditions and different camera pose, leading to smaller apple objects;

- The third environment highlights once more the effect of small object detection, with non-aligned average counting precision. Despite this, the introduction of the re-training improved the precision by almost 50% over the non-trained network, showing the positive effects of using dataset based on real apple orchards. The problems depicted in chapter 5.2.2 are still valid, mainly related to the camera pose;

Still, the real video results can only be taken as indicatives, as stated before because of human error in deriving ground truth and because some of the detections can be affected by double counts errors. The third environment The inference time for each segment has been on average 11.07 FPS, similar to the previous cases.

5.3.4 Standard and trained network comparison

The comparison between the two cases highlights some improvements in the overall performance of the network. The most important improvements are related to False Positives and False Negatives results, which respectively have seen a 25.3% and a 8.14% improvement, mainly due to the custom dataset built and implemented in the training, yielding to precision and recall gains of respectively 0.41% and 0.65%. The output count of the system is a reliable measure of the number apples present in the orchard, with an error due to double counts of about 2% in the standard network and 1.5% in the retrained case.

Averaged results comparison			
Metric	Standard	Re-trained	△%
Simulation avg FP	10.7	8.0	**-25.3%**
Simulation avg FN	44.2	40.6	**-8.14%**
Simulation P	0.981	0.985	**+0.41%**
Simulation R	0.924	0.930	**+0.65%**
Simulation F1	0.951	0.957	**+0.63%**
Real Batch 1 Error	11.07%	9.46%	**-14.54%**
Real Batch 2 Error	10.06%	8.16%	**-18.89%**
Real Batch 3 Error	27.81%	14.10%	**-49.30%**

Table 5.9. Performance comparison between the standard YOLOv4 network and the re-trained network

Training the network also with heavy occluded objects is hypothesised as the main factor that lead to improvements in False Positives and False Negatives. Having a consistent dataset, built also taking care of object occlusion directly in the training process, has benefited the overall performance of the system. This also is reflected by the augmented precision over real-world videos, where the training dataset sees the highest match with the input data. In particular, the improvements in the third environment, the

most difficult one due to camera pose and apple density, are notable, with almost 50% improvement over the standard network. A brief comparison is highlighted in the table 5.9.

Chapter 6

Conclusions and future work

The main goal of the thesis was do develop a system able to perform a fruit counting task in an orchard, with particular reference to apple fruits. This problem has lead to develop a detector and tracker based on the YOLO architecture, able to obtain counting accuracy of the fruits in the range of 1.5% to 2%, depending on lighting and working conditions, while featuring a counting error over the whole number of fruits present in the orchard ranging from 5% to 12%. In order to evaluate the system performance, several simulations and tests on real world videos have performed, in various working conditions. The first conclusion to be made about the work is that it strongly highlighted the importance of simulation of such a system in a controlled environment where the main variables such as ground truth references, light conditions and positions are known and controllable. This allows the developer to have a baseline reference to work with and evaluate the impact on the final performance of further modules introduced. Moreover, a testing environment such as this one can be interesting to look for the best working conditions for a system or simulate custom working conditions such as mid-night. The last benefit of using simulations is the large economical benefits in terms of testing costs.

As for the real tests, they depicted a situation close to reality, with some performance worsening due to working conditions not completely adherent to the simulated setup. However the tests, while showing satisfactory precision figures, highlighted low variance in the produced outputs, underlining a general consistency of the proposed solution. The conducted tests have proven to be very useful to indicate the weak points and the problem faced

by the system, summarised in the following paragraphs.
The first issue is represented by small objects compared to the viewport, such as far away o small fruits. This issue can be addressed easily in two ways. The first is to pay particular attention to camera position with respect to the crop, while also taking care of the camera lenses. The objective is to have the fruit seen by the image as the biggest it can possibly be. The other solution is represented by increasing the neural network input size, penalising the system inference speed. Depending on the usage of such a system, this can be a problem if it must deployed in application *at the edge*, seen in precision agricolture. The latter feature real-time application done on-site, with low power hardware that consequently cannot whitstand high computational power tasks such as high resolution computer vision tasks. While this problem can be partially addressed with dedicated platform such as NVidia Xavier or NVidia Jetson or TPUs (Tensor Processing Units, hardware accelerators developed specifically to improve AI neural network performance), it is a common best practice to have a neural network model input size the lower (or lighter in computational terms) it can possibly be. An alternative to be considered for further works that is expected to output even better results, is to employ a custom dataset built exactly on the needs of the current system and built specifically for it, thus done by employing the same camera pose descripted in the present work. This work highlights in fact that specific datasets can provide interesting improvements with respect to the baseline solution. In this perspective, a combination of a custom dataset, a dedicated hardware and some fine-tuning, can possibly make possible fruit counting at the edge with satisfactory results.

Detector and tracker parameter sensitivity is another point of the system that needs to be enhanced in future version of the work. In fact, while the detailed setup have proven to work efficiently and consistently in various working conditions, it has been clear that every video has its own best set of tuning parameters, both in terms of surrounding environment aspect such as lighting and camera pose and in terms of fruit-related features, such as fruit type and maturity. The best solutions to improve the robustness depends on the final applications of the system. Self-adjusting system parameters based on lighting conditions can be a solution, but while the development of such a system is non-trivial, it must be also validated trough multiple testing, as seen in this work. The best solution to obtain consistent results to be utilised in further processes, would be tuning the machine in order to work with well-known fruits, in well-known working conditions, such as mid-day as highlighted in the present work.

A further way to enhance the present system is integrate a real-time identity switch detector, a system able to assess whether an object have already been detected and tracked, thus correcting the final count. This can be done in a few ways. Deep learning o machine learning implementations that allow to recognise previously tracked objects are theoretically a solution, but the practical implementation is non-trivial by the fact that fruit objects do not have substantially different visual features to work with, thus making tracking algorithms such as *DeepSORT* produce even worse performance if compared to the standard *SORT*. The best way to tackle the problem is to work with more refined apple trajectory estimation systems, based on camera motion data, fruit size and depth estimation. In particular, with SfM techniques is possible to estimate the depth of every detected fruit starting from the RGB image and thus build a virtual 3d map of the orchard, in which the double counts are easily spotted since being overlapping objects.
The last point of improvement that can be done to the work is related to the simulated world. In fact, the latter can be further improved with features such as multi-row apple orchards, introduction of realistic 3d background and fruit object variance in terms of texture colour. Better adherence to reality can also be achieved in the apple location randomisation process.

Future work can use this as a baseline to develop systems able to improve the output results and improve the overall robustness. A way to do this, as detailed in the previous paragraph, is by introducing SfM RGB depth estimation for 3d-mapping of the orchard, allowing the system to easily recognise double counts but more importantly to have the baseline for very important data-aggregation features. In fact, having a realiable 3d map of the orchard, the machine can be added with further machine learning based modules able to derive properties such as fruit maturity stage, fruit size, health state or hygrometric level, exploiting the concept of *precision agricolture*. The agricultural operator can thus make meaningful decisions about the crop management looking at the aggregated data, optimising time and resources. A further implementation of the system can be devoted to the navigation part, making the robot able to navigate autonomously in SLAM (Simultaneous localisation and mapping) on the orchard or execute a pre-planned route through satellite imaging and GPS location.

Bibliography

[1] Frederic B. Fitch. "Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133." In: *Journal of Symbolic Logic* 9.2 (1944), pp. 49–50. DOI: 10.2307/2268029.

[2] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[3] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: https://books.google.it/books?id=EoYBngEACAAJ.

[4] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[5] Stamatia Dasiopoulou et al. "Knowledge-assisted semantic video object detection". In: *Circuits and Systems for Video Technology, IEEE Transactions on* 15 (Nov. 2005), pp. 1210–1224. DOI: 10.1109/TCSVT.2005.854238.

[6] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1904–1916. DOI: 10.1109/TPAMI.2015.2389824.

[7] Yann LeCun, Y. Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[8] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[9] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[10] ISO/TC 108/SC 2. *Mechanical vibration — Road surface profiles — Reporting of measured data*. Nov. 2016.

[11] Alex Bewley et al. "Simple Online and Realtime Tracking". In: *CoRR* abs/1602.00763 (2016). arXiv: 1602.00763. URL: http://arxiv.org/abs/1602.00763.

[12] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[13] Steven W Chen et al. "Counting apples and oranges with deep learning: A data-driven approach". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 781–788.

[14] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *CoRR* abs/1703.07402 (2017). arXiv: 1703.07402. URL: http://arxiv.org/abs/1703.07402.

[15] Thomas DH Jarvinen et al. "Multiple Object Tracking-by-Detection for Fruit Counting on an Apple Tree Canopy". In: *2018 ASABE Annual International Meeting*. American Society of Agricultural and Biological Engineers. 2018, p. 1.

[16] Shu Liu et al. "Path Aggregation Network for Instance Segmentation". In: *CoRR* abs/1803.01534 (2018). arXiv: 1803.01534. URL: http://arxiv.org/abs/1803.01534.

[17] Xu Liu et al. "Robust Fruit Counting: Combining Deep Learning, Tracking, and Structure from Motion". In: *CoRR* abs/1804.00307 (2018). arXiv: 1804.00307. URL: http://arxiv.org/abs/1804.00307.

[18] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[19] Sanghyun Woo et al. "CBAM: Convolutional Block Attention Module". In: *CoRR* abs/1807.06521 (2018). arXiv: 1807.06521. URL: http://arxiv.org/abs/1807.06521.

[20] Debora Cravero. "Apples recognition on trees with neural networks". 2019.

[21] Nicolai Häni, Pravakar Roy, and Volkan Isler. "A comparative study of fruit detection and counting methods for yield mapping in apple orchards". In: *Journal of Field Robotics* 37 (Aug. 2019). DOI: 10.1002/rob.21902.

[22] Anand Koirala et al. "Deep learning for real-time fruit detection and orchard fruit load estimation: benchmarking of 'MangoYOLO'". In: *Precision Agriculture* 20 (Dec. 2019). DOI: 10.1007/s11119-019-09642-0.

[23] Chien-Yao Wang et al. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *CoRR* abs/1911.11929 (2019). arXiv: 1911.11929. URL: http://arxiv.org/abs/1911.11929.

[24] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[25] Shuqin Tu et al. "Passion fruit detection and counting based on multiple scale faster R-CNN using RGB-D images". In: *Precision Agriculture* (2020), pp. 1–20.

[26] Kenta Itakura et al. "Automatic pear and apple detection by videos using deep learning and a Kalman filter". In: *OSA Continuum* 4.5 (2021), pp. 1688–1695.

[27] Mubashiru Olarewaju Lawal. "Tomato detection based on modified YOLOv3 framework". In: *Scientific Reports* 11.1 (2021), pp. 1–11.

[28] Guojin Li et al. "Lemon-YOLO: An efficient object detection method for lemons in the natural environment". In: *IET Image Processing* (2021).

[29] *Google Colab*. URL: https://colab.research.google.com/.

[30] hunglc007. *tensorflow-yolov4-tflite*. https://github.com/hunglc007/tensorflow-yolov4-tflite.

[31] MaximeHerpin. *mudlar$_t$ree*. https://github.com/MaximeHerpin/modular_tree/tree/blender_28.

[32] pjreddie. *darknet*. https://github.com/pjreddie/darknet.

[33] *Roboflow - Give your software the sense of sight*. URL: https://roboflow.com/.

[34] Tim Searchinger Janet Ranganathan Richard Waite and Craig Hanson. *How to Sustainably Feed 10 Billion People by 2050, in 21 Charts*. 2018. URL: https://www.wri.org/insights/how-sustainably-feed-10-billion-people-2050-21-charts.