University of study of Trento

Department of Industrial Engineering

Master of Science in Mechatronics Engineering

Embedded Systems

Enhancing UAV capabilities with machine learning on board

Supervisor

Prof. Brunelli Davide

Candidate

Francesco Argentieri

ID 183892

Academic Year 2018/2019

Abstract

The aim of this thesis is to increase the capabilities of an Unmanned Aerial Vehicles (UAVs) using convolutional neural networks for classification and object detection from images captured by the camera. In recent times, the use of UAVs has increased both in the military and civilian fields, such as for example in traffic control, surveillance, deliveries, photography and exploration. Within the UAV family, helicopters are preferred to fixed-wing aircraft especially for their vertical take-off and landing capacity. These are used in activities where environments and circumstances are dangerous to humans. In parallel successes of deep learning techniques in solving many complex tasks such as planning, localization, control and perception by starting from sensor data acquired in real environments make it suitable in autonomous robotic applications.
The development of dedicated hardware and software such as CUDA for Nvidia GPUs which shows best performances in computer vision, speech recognition, signal processing applications and so on.
A further aspect taken into consideration is the continuous evolution in the field of Information Technology (IT) driven by the continual request for decreasing capacity and costs. So embedded systems represent important slice of the entire IT sector, in particular they guide thanks to their mutual dependence on hardware and software in the mobile field and the Internet of Things (IoT).
These devices take advantage of application characteristics to optimize the design. Thus the introduction of platforms dedicated to neural computation has pushed the adoption of specific energy efficient and powerful processors.

This project providing the drone with greater awareness of the environment that surrounds it, making it less dependent on the pilot and therefore more autonomous in fulfilling set tasks. Although commercial solutions already implemented in drones available on the market, these are distributed in closed form. Hence the idea of expanding the drone's capabilities through open source software.
The project to achieve the objectives is divided into several steps which are not independent and which constrain design choices. The first problem is the need to classify and

determine the position of a target within the image. This is solved by the use of deep neural networks, i.e. convolutional neural networks. In particular, the construction of the training dataset affects the response provided by the neural network, so a dataset for the task will guarantee better results. Therefore through the use of 3D graphics techniques, a dataset was built based on plausibly real scenarios and on the instruments mounted on board the drone such as cameras.
The training process based on fine-tuning techniques involves adjusting the parameters in order to ensure a good compromise between calculation times and accuracy of the result. The creation of ad-hoc software capable of being executed on multiple platforms is a requirement in the prototyping phase. For this the use of a framework is essential for the success of this aspect. Furthermore, it is necessary to guarantee communication between the devices for this reason and being in an initial phase, it was preferred the TCP protocol which guarantees the control and reception of packets sent within the computer network.
In the use of both color and thermal cameras, it is necessary that the acquisition of the video streams proceeds without blocking during the execution of the program using parallelization. The processing of the neural network on general-purpose processors represents a high computational load which negatively affects the time required to obtain the response. Therefore the use of a framework capable of guaranteeing a compression of the neural model response in extremely short times proves to be a winning choice also for hardware dedicated to mobile and IoT products.
A brief introduction to the world of UAVs is given in chapter 1 where the general characteristics of these approaches are presented with a brief representation of their constituent components and the autonomy levels that can be implemented in this type of aircraft.
Chapter 2 introduces the hardware used to make the prototype by analyzing the computer boards in particular the Raspberry Pi 3b and the Google Coral Dev-Board.
Although these are similar they differ in the mounted processor therefore if we use the first board we find an ARM cortex-A53 32-bit processor whereas for the second board we find a processor of the same class but with 64-bit processing flanked by a Tensor Processor Unit for neural calculus.
There are also cameras used to acquire images, in particular there is the Raspberry Pi-Camera V2 equipped with sensor 8-Mega-pixels. Instead, for the acquisition of thermal images, a FLIR Lepton 2.5 with a sensor capable of providing images of the size 80×60 was preferred. Finally, it is presented in the logic that makes the use of the Tensor Processor Unit so attractive and its implications in energy consumption and tensor calculation.
In chapter 3 the software developed to be executed on the boards is analyzed.
In particular, they justify the choices taken to use the Qt framework in order to guaran-

tee ease during the development phase and cross-platform support. The critical functions within the program are analyzed in detail, highlighting the design and implementation choices. As well as a comparison of the performances between the tested architectures, in particular: Intel i7 (x86_64), ARM cortex-A53 (armv7l) and ARM cortex-A53 (aarch64).
The design and implementation of the dataset is presented in chapter 4, this represents a fundamental aspect in the training process of the neural network. In detail, the organization of the dataset is discussed and how it should be implemented to guarantee the solution of the object detection problem. In particular, the annotation work carried out on the images and how they are then processed by the neural network. The choice of the neural network to be trained is discussed in depth by carrying out a trade-off between performances, that is, having the shortest execution time to effect the inference. Correctness in the suggested answer to the classification and object detection problem. Without forgetting the possibility of being performed on specific hardware.
Chapter 5 presents the results achieved in the execution of the project highlighting the strengths of the choices made and the solutions used. In contrast, there are the limits of the model used, the quantization process as well as the choices partly due to the still immature technology and the lack of references in the literature.
The last chapter 6 presents possible future developments, strategies that will benefit the various aspects examined in the thesis. In particular, diversify and characterize the dataset in order to make the classification and object detection even more characteristic and specific. Adopting different and more optimized neural network models. As well as different training techniques and tools. Lastly, a more efficient and efficient optimization of the code to limit and reduce time and consumption of resources and energy for the benefit of greater execution speed and downtime with implications also on the energy aspect.

Contents

List of Figures

List of Tables

Chapter 1

Intoduction

The number of Unmanned aerial vehicles (UAVs) is increasing and have become an attractive in recent years.[1, 2] Studies are in progress on UAVs to utilize as an ideal platform for civilian tasks or military tasks, such as inspection, delivery, reconnaissance, or surveillance like the mobile robots based on autonomous navigation, real-time path planning, and object recognition.[3, 4] Meanwhile, UAVs have some challenges for autonomous flight, such as control strategy including parameter tuning, adaptive control, real-time path planning, and object recognition under uncertain environments. However, these approaches still had difficulties with sensors, system dynamics, qualities, and so on. In recent years, machine learning has become an attractive approach to overcome these challenges in UAVs for autonomous flight. Machine learning enables UAVs to recognize patterns or predict them from data without designed programming for autonomous flight.[5] The project's focus on machine learning conjunction autonomous flight, including control strategies and object recognition.

1.1 Unmanned aerial vehicle

An unmanned aerial vehicle UAV (commonly known as a drone) is an aircraft without a human pilot on board and a type of unmanned vehicle. UAVs are a component of an unmanned aircraft system (UAS); which include a UAV, a ground-based controller, and a system of communications between the two.

The flight of UAVs may operate with various degrees of autonomy: either under remote control by a human operator or autonomously by on-board computers. Compared to crewed aircraft, UAVs were originally used for missions too dangerous for humans.[6]

While they originated mostly in military applications, their use is rapidly expanding to commercial, scientific, recreational, agricultural, policing and surveillance, product deliveries, aerial photography, infrastructure inspections, and drone racing.[7]

(a) *Northrop Grumman Bat carrying EO/IR and SAR sensors, laser range finders, laser designators, Infra-Red cameras.*

(b) *A DeltaQuad VTOL fixed wing surveillance UAV.*

Figure 1.1: UAVs used in various operations.
Source: Wikipedia

1.1.1 UAV components

Crewed and un-crewed aircraft of the same type generally have recognizably similar physical components. One of the differences is the absence of the cockpit and environmental control system or life support systems. Some UAVs carry payloads such as a camera or other kinds sensors smaller and lightweight. Small UAVs have assumed a characteristic quad-copter design particular recognizable, although other scheme are realizable. With the continuous development in the introduction of new parts or revisited ones the process of miniaturized require less-power propulsion and increase battery runtime.
Control systems for UAVs are different for remote human control, a camera and video link almost always replace the cockpit windows; radio-transmitted digital commands replace physical cockpit controls. Autopilot software is used on both crewed and uncrewed aircraft, with varying feature sets.[7]

Body UAVs assume different configuration based on requirement of task they perform for example aerial video shooting, surveys, territorial control and many more. Thus they

(a) *typical quadcopter design.*

(b) *multirotor drone design.*

Figure 1.2: Example of design choices in the realization of UAVs shape
Source: DJI; Italdron

are normally equipped with 4, 6, or 8 motor called quadcopter, exacopter or octocopter. The mainly difference that exist between different configuration is the payload capacity that it can carry in flight.

Power supply Small UAVs mostly use lithium-polymer batteries (Li-Po), while larger vehicles rely on conventional airplane engines. Scale or size of aircraft is not the defining or limiting characteristic of energy supply for a UAV. Battery elimination circuitry (BEC) is used to centralize power distribution and often harbors a microcontroller unit (MCU). Costlier switching BECs diminish heating on the platform.[7]

Hardware Systems mounted on the drone have undergone an evolution in step with the advancement of the IT sector, so much so that the analog controls have been gradually replaced by microcontrollers up to System On Chip (SoC) with single board cmaputer such as the Raspberry. In addition, UAVs are equipped with flight controllers, flight

controller boards and autopilots.

Sensors and Actuators Drone control is allowed through the cooperation of proprioceptive and exteroceptive sensors that send their information to the central processing unit. The data coming from the platform measures accelerations and angular speeds through accelerometers and gyroscopes that process and transmit the attitude data such as orientation and position. The GPS allows the spatial location of the aircraft on a map and to control the route with respect to a planned trajectory. The altimeter allows you to continuously record changes in altitude. The magnetometer is a device that allows you to define the magnetic field vector at the point where you measure with the drone and then obtain the orientation with respect to the North. Knowledge and control of these measures allows the central system to allow control of the actuators. Other more less common sensors may be present to perform the most varied tasks.

Communications Most UAVs use a radio for remote control and exchange of video and other data. The connections depending on the use cases and needs can be narrowband and broadband. Generally, the sending and receiving of the commands is carried out by means of a radio connection from the ground, especially for remote piloting.
Other connections that exploit protocols such as TCP/IP are used for sending multimedia data such as filming areas and are transmitted to mobile devices such as smartphones, tablets and so on. Other types of connections can be used depending on the sector of use, in fact for the military sector these may differ to ensure greater safety and robustness.
More and more UAVs are implementing the MAVlink protocol for the transport of data of control and control between piloting on the ground and aircraft.

1.1.2 Autonomy

UAVs have various levels of autonomy, that is, they are not completely autonomous and do not require continuous intervention by the remote operator. They have algorithms of return to the house which can be performed with a certain level of autonomy. Up to high levels of autonomy which allow more complex operations. Autonomy derives from the awareness of the situation in which the drone is located. The knowledge derives from the data acquired by the sensors mounted on board the drone. The use of sensor fusion to integrate the information collected allows to limit the error committed and to increase the precision and accuracy of the information obtained. UAV's degrees of autonomy are often implemented by UAV manufacturers often build in specific autonomous operations, such as:

- Self-level: attitude stabilization on the pitch and roll axes.

- Altitude hold: The aircraft maintains its altitude using barometric or ground sensors.
- Hover/position hold: Keep level pitch and roll, stable yaw heading and altitude while maintaining position using GNSS or inertal sensors.
- Headless mode: Pitch control relative to the position of the pilot rather than relative to the vehicle's axes.
- Care-free: automatic roll and yaw control while moving horizontally.
- Take-off and landing (using a variety of aircraft or ground-based sensors and systems; see also:Autoland)
- Failsafe: automatic landing or return-to-home upon loss of control signal.
- Return-to-home: Fly back to the point of takeoff (often gaining altitude first to avoid possible intervening obstructions such as trees or buildings).
- Follow-me: Maintain relative position to a moving pilot or other object using GNSS, image recognition or homing beacon.
- GPS waypoint navigation: Using GNSS to navigate to an intermediate location on a travel path.
- Orbit around an object: Similar to Follow-me but continuously circle a target.
- Pre-programmed aerobatics (such as rolls and loops).[7]

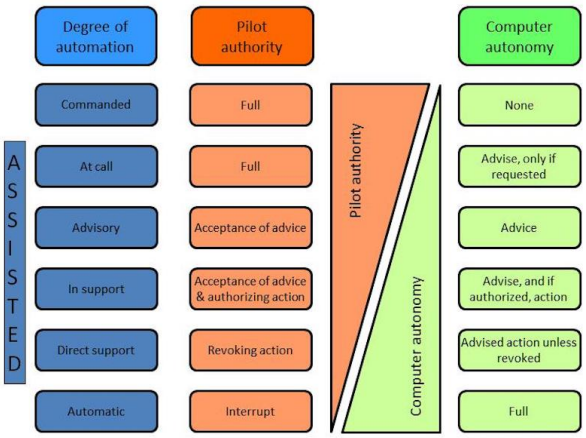


Figure 1.3: UAV's degrees of autonomy.
Source: Wikipedia

Chapter 2

Hardware

This chapter describes the hardware used to realize out the project. Where the use of the Raspberry Pi single board computer designed to have high performance both in education and science coupled with 8 mega-pixel camera module CMOS sensor was an almost conditioned choice. Shows that scientific and engineering–grade imagery can be produced with the Raspberry Pi 3 and its V2.1 camera module. It also presents a brief introduction to thermography and its physical principles before introducing Lepton 2.5 thermal camera module, made by the company FLIR which is currently one of the top manufactures of thermal camera solutions. The Coral Dev Board is a single-board computer that contains an Edge TPU coprocessor. It's ideal for prototyping new projects that demand fast on-device inferencing for machine learning models. The Coral Dev Board is ideal when you need to perform fast machine learning (ML) inferencing in a small form factor.

2.1 Raspberry Pi 3

The Raspberry Pi is a high-performance single-board computer designed to experiment and solve real-world problems. This small computer supports a camera module that uses a Sony IMX219 8 mega-pixel CMOS sensor.

2.1.1 Specification

As of early 2016, over 8 million Raspberry Pi's had been sold, making it one of the most popular single-board computers on the market.[8]
The Raspberry Pi credit-card-sized computer supports several accessories, including a camera module containing the Sony IMX219 sensor. This computer and camera configuration is of particular interest since it can provide raw-data format imagery that can

be used for a multitude of applications, including computer vision, biophotonics, medical testing, remote sensing, astronomy, improved image quality, high dynamic range (HDR) imaging, and security monitoring. The Raspberry Pi 3 is the third generation single board Raspberry Pi computer and became available to consumers in February 2016.
Some of the more significant Raspberry Pi attributes, including interfaces, are described in Table (2.1). The Raspberry Pi Foundation provides several operating systems for the Raspberry Pi 3, including Raspbian and a Debian-based Linux distribution, as well as third-party Ubuntu, Windows 10 IOT Core, RISC OS, and specialized distributions for download.[9]

Table 2.1: Raspberry Pi 3 Specification.

CPU 1.2 GHz 64-bit ARM Cortex-A53
1 GB of RAM LPDDR2 (900 MHz)
Wireless N and Blue-tooth 4.1 communication
Four USB ports
HDMI interface
Ethernet port
MicroSD card slot
40 GPIO pins
Camera interface
Composite video audio jack

2.2 Raspberry Pi, Camera Board V2

The camera is based on the Sony IMX219 silicon CMOS back-lit sensor and produces 8 mega-pixel images that are 3280 × 2464 pixels in size. The IMX219 sensor operates in the visible spectral range from 400 to 700 nm).[10] Sensor specifications are detailed in Table (2.2).

2.2.1 Specification

The V2 camera module operates at a fixed focal length (3.04 mm) and single f-number (F2.0) typically focused from the near-field to infinity. Images can be captured at ISO settings between 100 and 800 in manually set increments of 100 and camera exposure times between 9 µs and 6 s using a rolling shutter. Some of the more significant camera specifications are shown in Table 2.3. In addition to still photos, the Raspberry Pi Sony IMX219 sensor supports a cropped 1080p format at 30 frames per second (fps) and full-frame imaging video at up to 15 fps, but not in raw-data format. The entire camera board is small 25 mm × 25 mm × 9 mm and weighing about 3 g.

Table 2.2: Sony IMX219 sensor chip specifications.

Sensor parameter	**Specification**
Image sensor type	Back-lit CMOS
Image size	Diagonal 4.60 mm (type 1/4.0)
Number of active pixels	3280 (H) × 2464 (V) ∼ 8.08 mega-pixels
Chip size	5.095 mm (H) × 4.930 mm(V) (w/ Scribe)
Unit cell size (pixel)	1.12 µm (H) × 1.12 µm(V)
Substrate material	Silicon
Bit depth	10-bit A/D converter on chip
Data output	CSI2 serial data output (selection of 4 lane/ 2 lane)
Communication	2-wire serial communication circuit on chip
Max full-frame frame rate	30 frames/s
Pixel rate	280 mega-pixel/s (all-pixels mode)
Data rate	Max. 755 Mbps/lane (at 4 lane), 912 Mbps / lane(at 2 lane)

It connects directly to the Raspberry Pi 3 through a 15 pin mobile industry processor interface (MIPI) camera serial interface and is shown alongside a Raspberry Pi 3 in Figure (2.1).[8, 11]

Table 2.3: Raspberry Pi camera Specification.

Camera parameter	**Specification**
Lens focal length	3.04 mm
f-number	2.0
Instantaneous field of view	0.368 mrad
Full-frame field of view	59.17 °(H) × 58.3 ° (V)

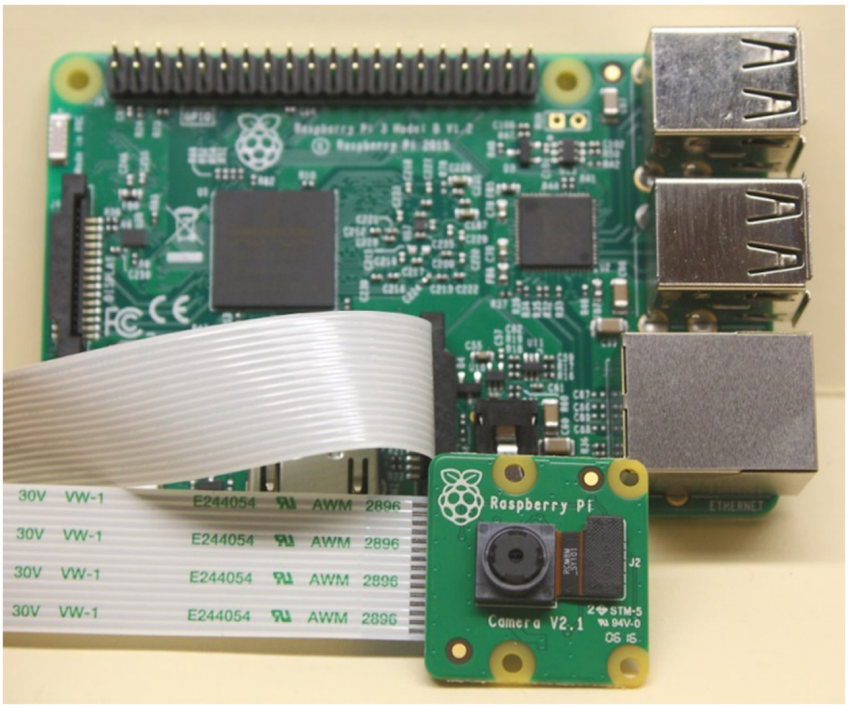


Figure 2.1: Raspberry Pi 3 and camera module V2.1.

2.3 Thermal imaging theory

The story of infra-red imaging started in 1800, when Herschel discovered infra-red radiation experimentally at long wavelengths just outside the visible spectrum of sun light. The quantitative explanation of incandescent infra-red radiation in 1900 by Max Planck started a development, which today has resulted in modern infra-red technologies with infra-red camera systems- These are also the result of scientific developments in semiconductor physics and micro-system technologies.[12]
Since its birth, it is possible to recognize three generations of infra-red cameras[13]: the first generation cameras were characterized by a single element detector, combined with two scanning mirrors to create infra-red images. Their main disadvantage was that they suffered from saturation problems. Saturation indicates the limit of the highest irradiation that can be measured by a detector. For digital sensors, since incident photoelectrons are converted in charges, each detector can store a maximum amount of charges known as the full well capacity.[12]
The second generation cameras were characterized by an increase in the number of detectors, positioned in a large linear array or in two small 2-D array. The third generation cameras, i.e., the ones currently used, are characterized by large focal plane array (FPA) detectors, thus increasing the reliability and sensitivity of such infra-red systems.[14]

2.3.1 Electromagnetic radiation

Electromagnetic radiation is all around (and within, and throughout) us and is comprised of everything from gamma radiation on the high frequency end to radio waves on the low frequency end. So the Figure (2.2) give an overview of EM waves, ordered according to their wave-length or frequency. This spectrum consists of a great variety of different waves. All of them can be observed in nature, and many have technical applications. Starting from the left of the figure, for example, γ-rays have the highest frequencies, that is, the shortest wavelengths.[15]

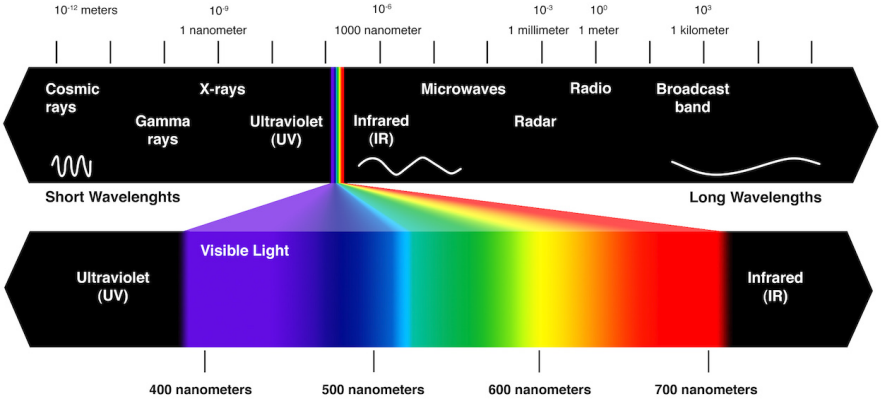


Figure 2.2: Composition of the spectrum of electromagnetic waves.
Source: https://socratic.org

The visible light, defined by the sensitive range of the light receptors in our eyes, only covers a very small range within this spectrum, with wavelengths from 380 to 780 nm. The adjacent spectral region with wavelengths from 780 nm up to 1 mm is usually called infra-red. This range is followed by microwaves, RADAR, and all EM waves that are used for radio, TV, and so on. While most imaging sensors detect radiation in the visible spectrum (wavelengths from 380 to 700 nm), long wave infra-red sensors detect radiation the infra-red spectrum, and it accounts for most of the thermal radiation emitted by objects near room temperature. Then for IR imaging, only a small range of the IR spectrum is used. It is shown in an expanded view in Figure (2.3).
Typically, three spectral ranges are defined for thermography: **long-wave (LW)** region from around 8 to 14 µm, **mid-wave (MW)** region from around 3 to 5 µm, **short-wave (SW)** region from 0.9 to 1.7 µm.
The origin of naturally occurring EM radiation is manifold. The most important phenom for thermography is the thermal radiation. In brief, the thermal radiation implies that every body or object at a temperature $T > 0$ K ($-273.15\,^{\circ}$C) emits EM radiation.

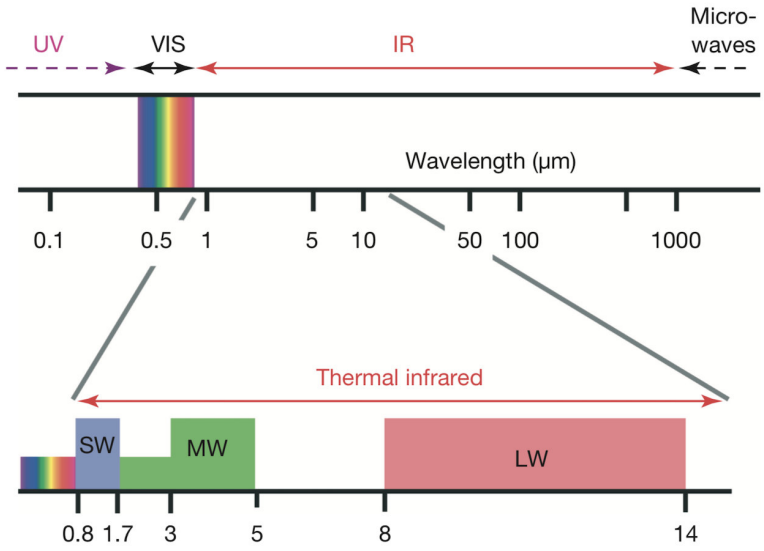


Figure 2.3: Infrared (IR) and adjacent spectral regions and expanded view of so-called thermal IR. This is the region where IR imaging systems for short-wave (SW), mid-wave (MW), and long- wave (LW) cameras exist. Special systems have extended ranges.
Source: [15]

The amount of radiation and its distribution as a function of wavelength depend on temperature and material properties.[15] For temperatures in the range of natural and technological processes, this radiation is in the thermal IR spectral region. This is known as the infra-red spectrum, and it accounts for most of the thermal radiation emitted by objects near room temperature.

2.3.2 Modern detectors

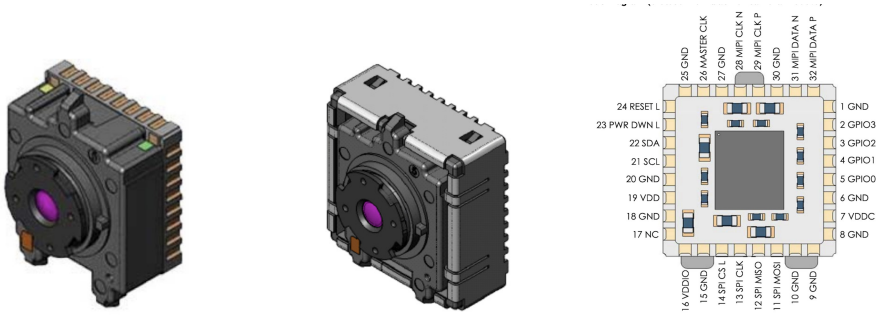
The main difference between a thermal imaging and normal image capturing is the sensor used. One type of the sensor used in thermal cameras is called a microbolometer. Essentially its functionality is similar to the CMOS or CCD sensor, just in different wavelengths. Unlike many other thermal sensor the microbolometer doesn't need an active cooling system to function a long period of times. Modern uncooled detectors use sensors whose working mechanism is based on a change of resistance, voltage or current when heated by IR radiation. Uncooled detectors are mostly composed by pyroelectric and ferroelectric materials or based on microbolometer technology. The thermal signal depends upon the radiant power but not upon its spectral content, i.e., it is wavelength independent.[12, 14]

2.4 Thermal Camera module Lepton 2.5

The thermal camera module we use in this project its made by FLIR, we will go through its technical specification, modes of capture and communication protocol for both video frame transfer and camera control. Information about the camera are extracted from official documentation.

2.4.1 Specification

Lepton is an infra-red camera system that integrates a fixed-focus lens assembly, an 80×60 long-wave infra-red (LWIR) micro-bolometer sensor array, and signal-processing electronics. Easy to integrate and operate, Lepton is intended for mobile devices as well as any application requiring very small footprint, very low power, and instant-on operation. Lepton can be operated in its default mode or configured into other modes through a command and control interface (CCI). The effective frame rate of the camera is only 8.6 Hz, however for our needs this is not a problem. The camera only requires low voltage supply and has small power consumption of around 140 mW.



(a) *with and without socket.* **(b)** *Pinout Diagram.*

Figure 2.4: Lepton Camera sketch.
Source: [16]

See table (2.4) for more specifications. For better manipulation with the camera module we use a breakout board figure (2.5) with a housing for the Lepton camera module. The breakout board provides better physical accessibility, improves heat dissipation and increases input voltage supply range to 3–5 V, as it has its own regulated power supply. This power supply provides the camera module with three necessary voltages: 1.2, 2.8 and 2.5–3.1 V. The breakout board also supplies the camera with master clock signal.[17] The camera uses two interfaces for communication:

- SPI for transferring video frames from the camera to a SPI master device.
- I^2C for receiving control commands from the I^2C master device.

Table 2.4: Thermal camera specification

	FLIR Lepton 2.5	
Resolution (h x w)	80 × 60	pixels
Spectral Range	8 to 14	µm
Horizontal Field of View	51	°
Thermal Sensitivity	< 50	mK
Frame Rate	< 9	Hz
Control Interface	I^2C	
Video Interface	SPI	
Promised Time to Image	< 0.5	s
Integral Shutter	yes	
Radiometry	14-bit pixel value	
Operating Power	∼150	mW

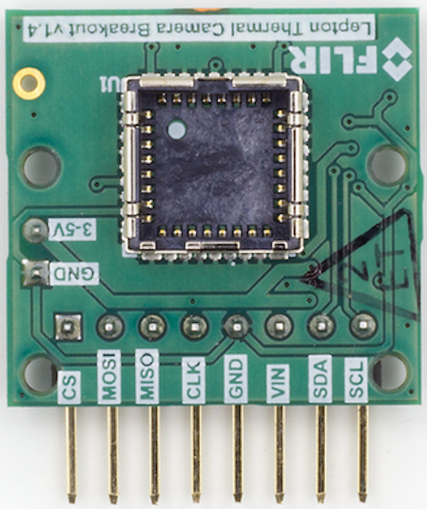


Figure 2.5: Breakout board.
Source: https://groupgets.com/manufacturers/getlab/products/flir-lepton-breakout-board-v1-4

2.4.2 System Architecture

The lens assembly focuses infrared radiation from the scene onto an 80 × 60 array of thermal detectors with 17-micron pitch. Each detector element is a vanadium-oxide (VOx) microbolometer whose temperature fluctuates in response to incident flux. The change in temperature causes a proportional change in each microbolometer's resistance. VOx provides a high temperature coefficient of resistance (TCR) and low $1/f$ noise,

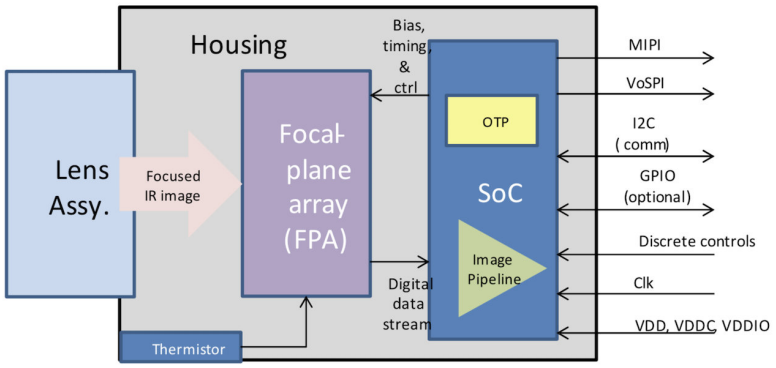


Figure 2.6: Lepton Architecture.
Source: [16]

resulting in excellent thermal sensitivity and stable uniformity. The microbolometer array is grown monolithically on top of a readout integrated circuit (ROIC) to comprise the complete focal plane array (FPA). For each frame, the ROIC senses the resistance of each detector by applying a bias voltage and integrating the resulting current for a finite period of time called the integration period. The serial stream from the FPA is received by a system on a chip (SoC) device, which provides signal processing and output formatting.

2.4.3 Video Pipeline

The video pipeline includes non-uniformity correction (NUC), defect replacement, spatial and temporal filtering, automatic gain correction (AGC), and colourization.

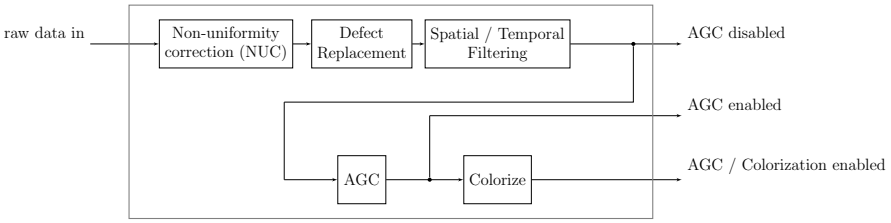


Figure 2.7: Lepton video pipeline block diagram.
Source: [16]

The non-uniformity correction (NUC) block applies correction terms to ensure that the camera produces a uniform output for each pixel when imaging a uniform thermal scene. Factory-calibrated terms are applied to compensate for temperature effects, pixel response variations, and lens-illumination roll-off. To compensate for temporal drift, the NUC block also applies an offset term that can be periodically updated at runtime via a

process called flat-field correction (FFC). The FFC process is further described in FFC States, (2.4.5).

The defect-replacement block substitutes for any pixels identified as defective during factory calibration or during runtime. The replacement algorithm assesses the values of neighboring pixels and calculates an optimum replacement value. The typical number of defective pixels is ≤ 1.

Temporal Filtering the image pipeline includes a number of sophisticated image filters designed to enhance signal-to-noise ratio (SNR) by eliminating temporal noise and residual non-uniformity. The filtering suite includes a scene-based non-uniformity correction (SBNUC) algorithm which relies on motion within the scene to isolate fixed pattern noise (FPN) from image content.

The AGC algorithm for converting the full-resolution (14-bit) thermal image into a contrast-enhanced image suitable for display is a histogram-based non-linear mapping function. See (2.4.6).

The colorize block takes the contrast-enhanced thermal image as input and generates a 24-bit RGB color output.

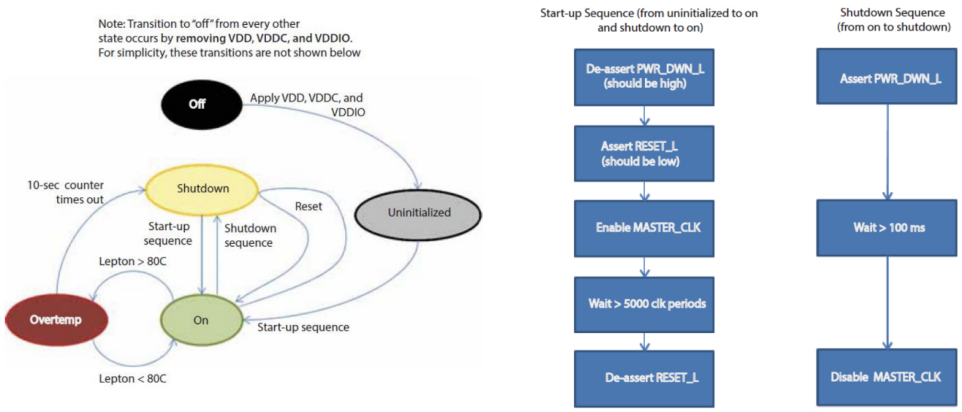
2.4.4 Power States

Lepton currently provides five power states. As depicted in the state diagram shown in Figure (2.8a), most of the transitions among the power states are the result of explicit action from the host. The automatic transition to and from the overtemp state is an exception. In the figure (2.8b), transitions that require specific host-side action are shown in bold. Automatic transitions are not bolded.

- Off: When no voltage is applied, Lepton is in the off state. In the off state, no camera functions are available.
- Uninitialized: In the uninitialized state, all voltage forms are applied, but Lepton has not yet been booted and is in an indeterminate state. It is not recommended to leave Lepton in this state as power is not optimized; it should instead be booted to the on-state (and then transitioned back to standby if imaging is not required).
- On: In the on state, all functions and interfaces are fully available.

- Standby: In the standby state, all voltage forms are applied, but power consumption is approximately 4 mW. In the standby state, no functions are available, but it is possible to transition to the on state via the start-up sequence. The shutdown sequence is the recommended transition back to the standby state. It is also possible to transition between standby and on states via software commands, as further defined in the software IDD.

- Overtemp: The overtemp state is automatically entered when the Lepton senses that its temperature has exceeded approximately 80 °C. Upon entering the overtemp state, Lepton enables a "*shutdown imminent*" status bit in the telemetry line and starts a 10 s counter. If the temperature of the Lepton falls below 80 °C before the counter times out, the "*shutdown imminent*" bit is cleared and the system transitions back to the on state. If the counter does time out, Lepton automatically transitions to the standby state.



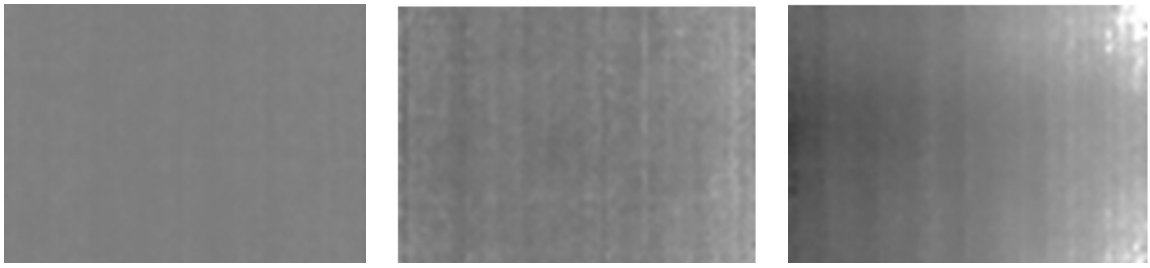
(a) *state diagram.* **(b)** *lepton Power Sequencing.*

Figure 2.8: Lepton Power Sequencing.
Source: [16]

2.4.5 FFC States

Lepton is factory calibrated to produce an output image that is highly uniform, such as shown in Figure (2.9a), when viewing a uniform-temperature scene. However, drift effects over long periods of time degrade uniformity, resulting in imagery which appears more grainy Figure (2.9b) and/or blotchy Figure (2.9c). Operation over a wide temperature range (for example, powering on at −10 °C and heating to 65 °C) will also have a detrimental effect on image quality. For scenarios in which there is ample scene movement, such as most handheld applications, Lepton is capable of automatically compensating

for drift effects using an internal algorithm called scene-based non-uniformity correction (scene-based NUC or SBNUC). However, for use cases in which the scene is essentially stationary, such as fixed-mount applications, scene-based NUC is less effective. In those applications, it is recommended to periodically perform a flat-field correction (FFC).
FFC is a process whereby the NUC terms applied by the camera's signal processing engine are automatically recalibrated to produce the most optimal image quality. The sensor is briefly exposed to a uniform thermal scene, and the camera updates the NUC terms to ensure uniform output. The entire FFC process takes less than a second.



(a) *Highly uniform image.*

(b) *Grainy image (high-spatial frequency noise).*

(c) *Blotchy image (low-spatial frequency noise).*

Figure 2.9: Examples of Good Uniformity, Graininess, and Blotchiness
Source: [16]

The current FFC state is provided through the telemetry line. There are three FFC states, as illustrated in Figure (2.10):

1. FFC not commanded (default): In this state, Lepton applies by default a set of factory-generated FFC terms.

2. FFC in progress: Lepton enters this state when FFC is commanded. The default FFC duration is nominally 23 frames.

3. FFC complete: Lepton automatically enters this state whenever FFC is completed. Lepton also provides an "FFC desired" flag in the telemetry line. The "FFC desired" flag is asserted at start-up, when a specified period (default = 3 minutes) has elapsed since the last FFC, or when the sensor temperature has changed by a specified value (default = 3 °C) since the last FFC. The "FFC desired" flag is intended to indicate to the host to command an FFC at the next possible opportunity.

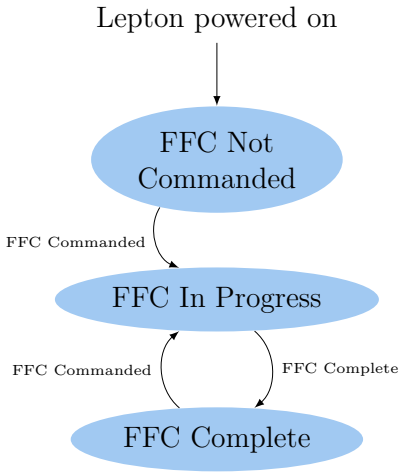


Figure 2.10: FFC States.

2.4.6 AGC Modes

There are two AGC modes:

- AGC disabled (default)
- AGC enabled

AGC is a process whereby the large dynamic range of the infrared sensor is collapsed to a range more appropriate for a display system. For Lepton, this is a 14-bit to 8-bit conversion.
In its most simplistic form, AGC can be a linear mapping from 14-bit to 8-bit; however, a simple linear AGC is generally incapable of providing pleasing imagery in all imaging conditions. For example when a scene includes both cold and hot regions (for example, a hot object in front of a cold background as illustrated in (2.12), linear AGC can produce an output image in which most pixels are mapped to either full black or full white with very little use of the gray shades (8-bit values) in between. Because of this limitation of linear AGC, a more sophisticated algorithm is preferred. Similar to most AGC algorithms that optimize the use of gray shades, Lepton's is histogram-based. Essentially a histogram counts the number of pixels in each frame that have a given 14-bit value. Figure (2.11) shows the concept for a 3×3 pixel area.

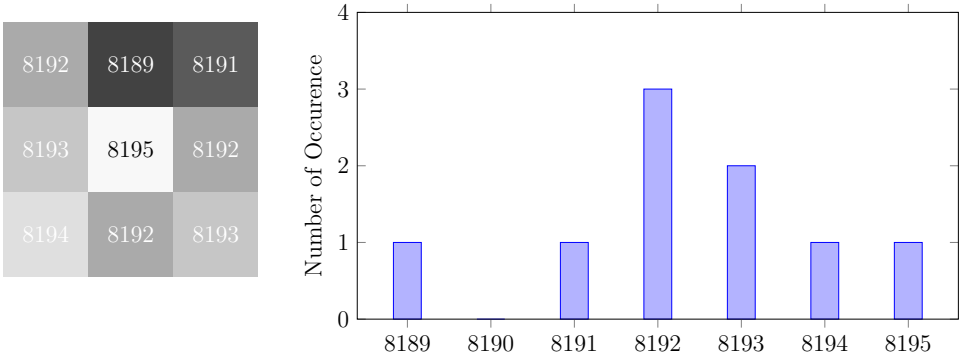


Figure 2.11: Illustration of a Histogram for a 3×3 Pixel Area.
Source: [16]

Classic histogram equalization uses the cumulative histogram as a mapping function between 14-bit and 8-bit. The intent is to devote the most gray shades to those portions of the input range occupied by the most pixels. For example, an image consisting of 60% sky devotes 60% of the available gray shades to the sky, leaving only 40% for the remainder of the image. By comparison, linear AGC "wastes" gray shades when there are gaps in the histogram, whereas classic histogram equalization allocates no gray shades to the gaps. This behaviour is in principle an efficient use of the available gray shades, but there are a few drawbacks:

- The resulting contrast between an object and a much colder (or hotter) background can be rendered poor by the fact the algorithm "collapses" the separation between such that the object is only one step gray shade above the background. This phenomenon is illustrated in (2.12).
- Too much emphasis can be placed on background clutter, particularly when a mostly isothermal background comprises a large fraction of the total image area. This is also illustrated in (2.12). The Lepton AGC algorithm is a modified version of classic histogram equalization that mitigates these shortcomings. One such modification is a parameter called "clip limit high". It clips the maximum population of any single bin, limiting the influence of heavily populated bins on the mapping function. Another parameter utilized by the Lepton algorithm is called "clip limit low". It adds a constant value to every non-zero bin in the histogram, resulting in additional contrast between portions of the histogram separated by gaps. Figure (2.12) is an example showing the benefit of the Lepton clip parameters.

A high value of clip limit high results in a mapping more like classic histogram equalization, whereas a low value results in mapping more like linear AGC. For clip limit low, the

(a) *Linear AGC.*

(b) *Classic Histogram Equalization.*

(c) *Lepton's Variant of HistogramEqualization.*

Figure 2.12: Comparison of Linear AGC and Classic–Lepton variant of Histogram Equalization
Source: [16]

opposite is true: a high value results in a mapping more like linear AGC, whereas a low value results in a mapping more like classic histogram equalization. The default values of both parameters produce a good compromise between the two; however, because optimum AGC is highly subjective and often application dependent, customers are encouraged to experiment to find settings most appropriate for the target application. By default, the histogram used to generate Lepton's 14-bit to 8-bit mapping function is collected from the full array. In some applications, it is desirable to have the AGC algorithm ignore a portion of the scene when collecting the histogram. For example, in some applications it may be beneficial to optimize the display to a region of interest (ROI) in the central portion of the image. When the AGC ROI is set to a subset of the full image, any scene content located outside of the ROI is not included in the histogram and therefore does not affect the mapping function (note: this does not mean the portion outside of the ROI is not displayed or that AGC is not applied there, only that those portions outside the AGC ROI do not influence the mapping function).[16]

2.5 Interface

The Raspberry Pi seen in section (2.1) has bi-directional I/O pins, which you can use to drive LEDs, spin motors, or read button presses. The board offers its GPIO over a standard male header on the board. Over the years the header has expanded from 26 pins to 40 pins while maintaining the original pinout. There are at least two, different numbering schemes you may encounter when referencing pin numbers:

1. Broadcom chip-specific pin numbers

2. `P1` physical pin numbers.

Here's a figure 2.13 showing all 26 pins on the `P1` header, including any special function they may have, and their dual numbers.

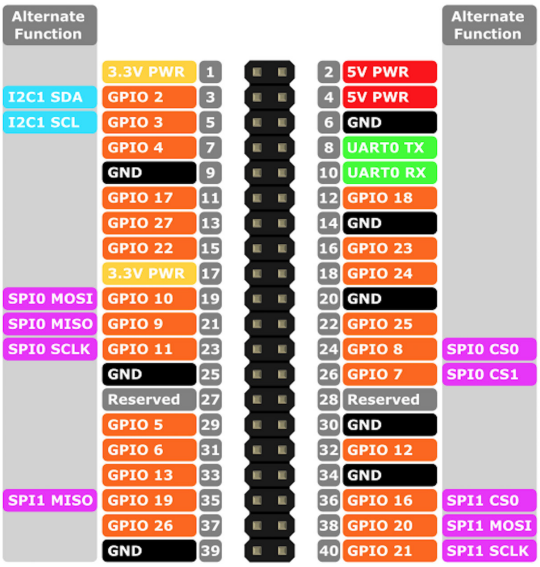


Figure 2.13: Raspberry Pi 3 Pin Mappings.
Source: Microsoft Windows Dev Center

The camera uses two interfaces for communication:

- **SPI** for transferring video frames from the camera to a SPI[1] master device.
- **I²C** for receiving control commands from the I²C[2] master device.

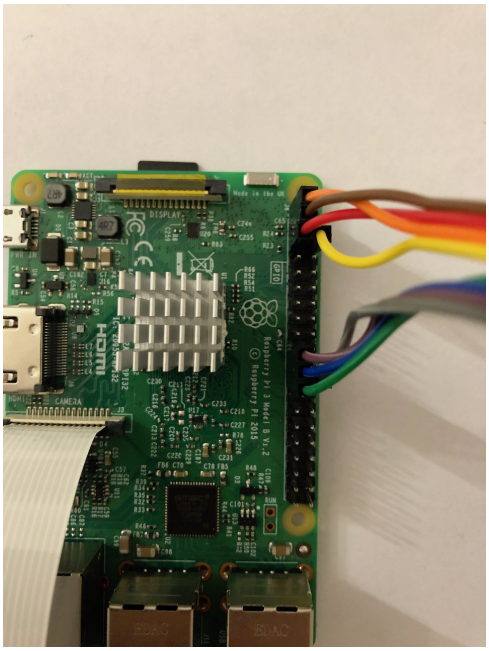
The Raspberry Pi's CPU has enough processing power to maintain smooth operation without delays, which turned out to be crucial for maintaining synchronization with the camera module when transferring video frames. Below is reported the connection scheme used between the FLIR breakout and the Raspberry Pi 's GPIO according to the figures (2.14) and the table (2.5).

[1]SPI – Serial peripheral interface bus
[2]I2C (Inter-integrated circuit)

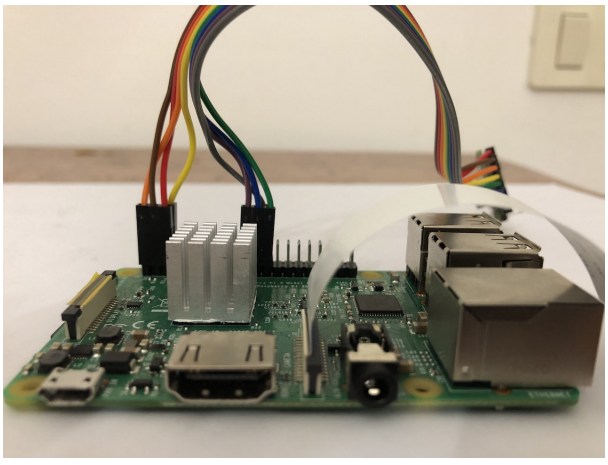
Raspberry GPIO	Breakout board	Alternative function	PIN
+3.3V	VIN	SPI0	1
SDA	SDA	SPI0	3
SCL	SCL	SPI0	5
GND	GND	SPI0	6
MOSI	MOSI	GND	19
MISO	MISO	3.3V	21
SCLK	CLK	I2C1	23
CE0	CS	I2C1	24

Table 2.5: Schematic connection GPIO

(a) *Breakout board detail.*



(b) *Top view.*



(c) *Frontal view.*

Figure 2.14: Realization of the connection between the Raspberry Pi 3 GPIO and Break out board

2.6 Coral Dev-Board

Nowadays some manufacturers replace the GPU for a TPU, a Tensor Processing Unit. Driven by the exponential interest for deep learning in different field such as research, industries, robotics. The Coral Dev Board is a single-board computer that contains an Edge TPU coprocessor. It's ideal for prototyping new projects that demand fast on-device inferencing for machine learning models. Coral Dev Board cannot train a neural network because the TPU is specific designed to work with special pre-compiled model and to be small and energy efficient.

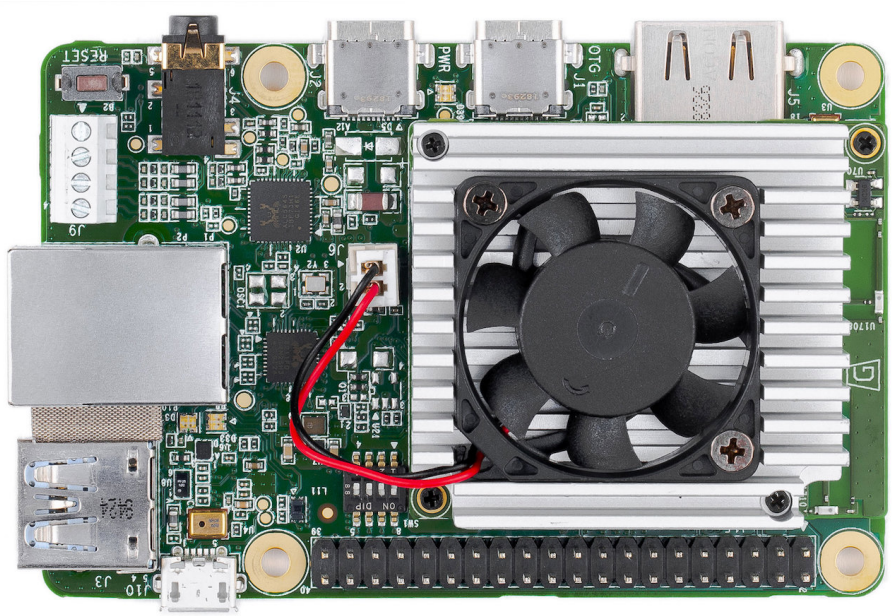


Figure 2.15: Coral Dev Board.
Source: https://coral.ai/docs/dev-board/get-started/

2.6.1 Overview

The Coral Dev Board is a single-board computer that's ideal when you need to perform fast machine learning (ML) inferencing in a small form factor. You can use the Dev Board to prototype your embedded system and then scale to production using the on-board Coral System-on-Module (SoM) combined with your custom PCB hardware. The SoM provides a fully-integrated system, including NXP's iMX 8M system-on-chip (SoC), eMMC memory, LPDDR4 RAM, Wi-Fi, and Bluetooth, but its unique power comes from Google's Edge TPU coprocessor. The Edge TPU is a small ASIC designed by Google that provides high performance ML inferencing with a low power cost. For example, it can execute state-of-the-art mobile vision models such as MobileNet v2 at almost 400 FPS, in a power efficient manner. The baseboard provides all the peripheral connections you

need to prototype a project, including USB 2.0/3.0 ports, DSI display interface, CSI-2 camera interface, Ethernet port, speaker terminals, and a 40-pin I/O header.
Key benefits of the Dev Board:

- High-speed and low-power ML inferencing (4 TOPS @ 2 W)
- A complete Linux system (running Mendel, a Debian derivative)
- Prototyping and evaluation board for the small Coral SoM (40 x 48 mm)

Edge TPU System-on-Module (SoM)
NXP i.MX 8M SoC (Quad-core Arm Cortex-A53, plus Cortex-M4F)
Google Edge TPU ML accelerator coprocessor
Cryptographic coprocessor
Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5 GHz)
Bluetooth 4.2
8 GB eMMC
1 GB LPDDR4
USB connections
USB Type-C power port (5 V DC)
USB 3.0 Type-C OTG port
USB 3.0 Type-A host port
USB 2.0 Micro-B serial console port
Audio connections
3.5 mm audio jack (CTIA compliant)
Digital PDM microphone (x2)
2.54 mm 4-pin terminal for stereo speakers
Video connections
HDMI 2.0a (full size)
39-pin FFC connector for MIPI DSI display (4-lane)
24-pin FFC connector for MIPI CSI-2 camera (4-lane)
MicroSD card slot
Gigabit Ethernet port
40-pin GPIO expansion header
Supports Mendel Linux (derivative of Debian)

Table 2.6: Coral Dev Board Features.

The Google Coral with a special TPU chip performing all tensor calculations works with special pre-compiled TensorFlow Lite networks. If the topology of the neural network and its required operations can be described in TensorFlow it may work well on the Google Coral.

2.6.2 8 bits integers

An alternative to reduce compute time is the use of integers 8-bit instead float 32-bit. The difference in the use of amount of memory allocated is reduced. The advantage derive from the Neural Network 's accuracy insensibility of number represented, while maintaining the same precision. On the basis of this we can deduce that will save memory, furthermore an cut-off of number of transistor in the chip because is not more necessary take into account floating point. Consequently the this permit to achieve processor powerful, small and energy efficient.

2.7 TPU explained in depth

The Google Coral has a TPU on board which speeds up the tensor calculations enormously. These tensor calculations are used in deep learning and neural networks.
The Google Coral Dev board use an ASIC made by the Google team called the Edge TPU. It is a much lighter version of the well-known TPUs used in Google's datacenter. It also consumes very little power, so it is ideal for small embedded systems. Nevertheless, the similarities in applied technology are significant.[18]

2.7.1 Neural Node

Neural networks used in deep learning consists of many neural nodes. They are all connected together in a defined way. The way these nodes are wired is called the topology of the network. This topology determines the function the network performs. See this list for a selection of several types of deep learning networks. Each node has always three basic components. A multiplier multiplies all the inputs with their respective so-called weight, the synapses. An adder who accumulates all the individual multiplications.
And an activation function that shapes the output given the addition. A schematic view below (2.16).

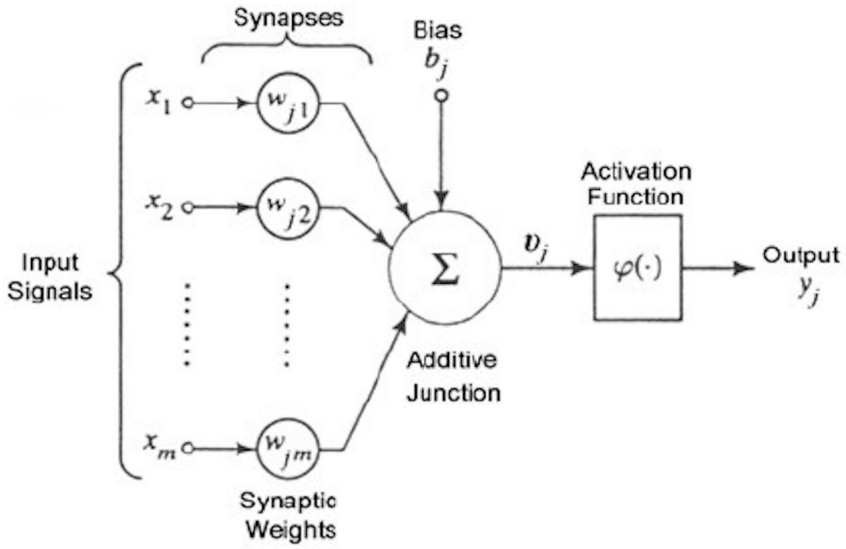


Figure 2.16: Artificial Neuron models and its parts.
Source: Adapted from Haykin (1994)

The formula can be written as follows (2.1).

$$y_i = \phi \left(\sum_{i=1}^{n} w_{ij} \cdot x_i \right) \tag{2.1}$$

Deep neural network is consist of millions of neural nodes distribute over many layers; despite the simplicity of the operation, only on multiplication and one addition, do you require longer time to obtain a result. Keep in mind that for training a network requires many epochs. In other words, the main problem is time. The algorithms is well suited for parallel execution. Although distributing the algorithm over the processes and threads can be an option actually provides disappoint results for the presence of bottlenecks due to architecture general purpose. On the other hand use of Graphical Processing Unit, the GPU on video card designed for efficient manipulation memory. Their highly parallel structure make them more efficient respect CPU especially for algorithms that process large blocks of data in parallel. The best choice is the use of the Tensor Processing Unit, the TPU. This device has been specially designed for the above neural node algorithm.

2.7.2 The adder

Generally a strategy adopted when software is tool slow is modify the hardware to reach the better achievements. The structure inside TPU is realized by three main components derived from neural node, then keeping in mind the scheme in figure (2.16): the **multiplier**, the **adder** and the **activation function** must be included in the hardware.
Start with analysing the diagram of the 4-bit adder realized in figure (2.17).

Where: `A` and `B` are the inputs. If the output overflows `C4` the carry out is set. `C0` is the carry-in of a previous phase.[18] Signals `A` and `B` propagate through the circuit and

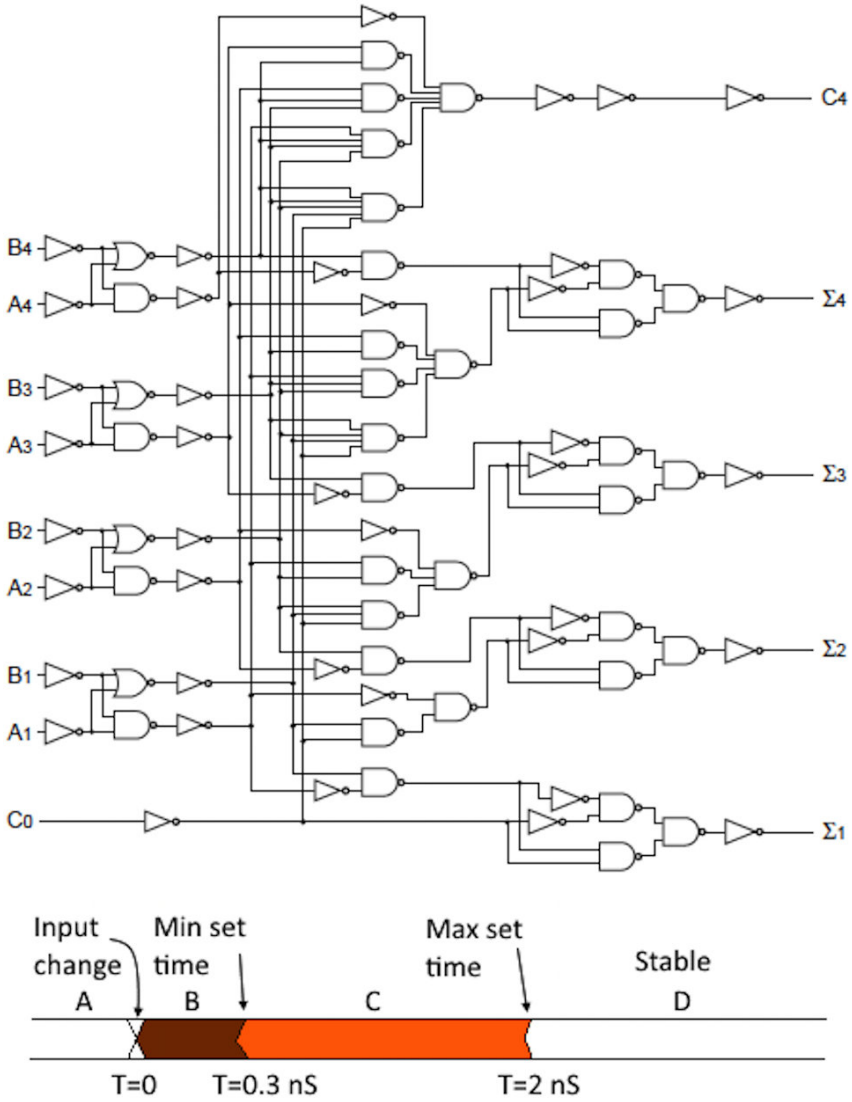


Figure 2.17: 4-bit adder
Source: Q-engineering

generate the result `A + B`. Changing one of them alters almost immediately the output. This happens extremely very fast, within a few nanoseconds. This propagation time depends on the number of digital ports whose output changes. The propagation time is therefore not fixed, but lies between two limits, a minimum and a maximum time, see diagram at the bottom of the drawing (2.17). By the way, al mentioned times are illustrative and have no relationship to any device.[18]

2.7.3 Pipeline

Consider the example where: the propagation time of one adder is 2 ns, therefore the maximum clock rate can be 500 MHz. Taking into account that a neural node can have many inputs, also hundreds, that must be accumulated this affect the propagation time that increase dramatically. This implies that the last adder in the chain must be attend all intermediate result before its output becomes stable. If we consider a chain of 250 input each of one with 2 ns delay, we obtain total time of 500 ns waiting.
Consequently clock results extremely slow 2 MHz, then solution adopted is structured pipeline where between each adder is inserted a memory element that keep keeps the result stable for the next adder. As show in figure (2.18).

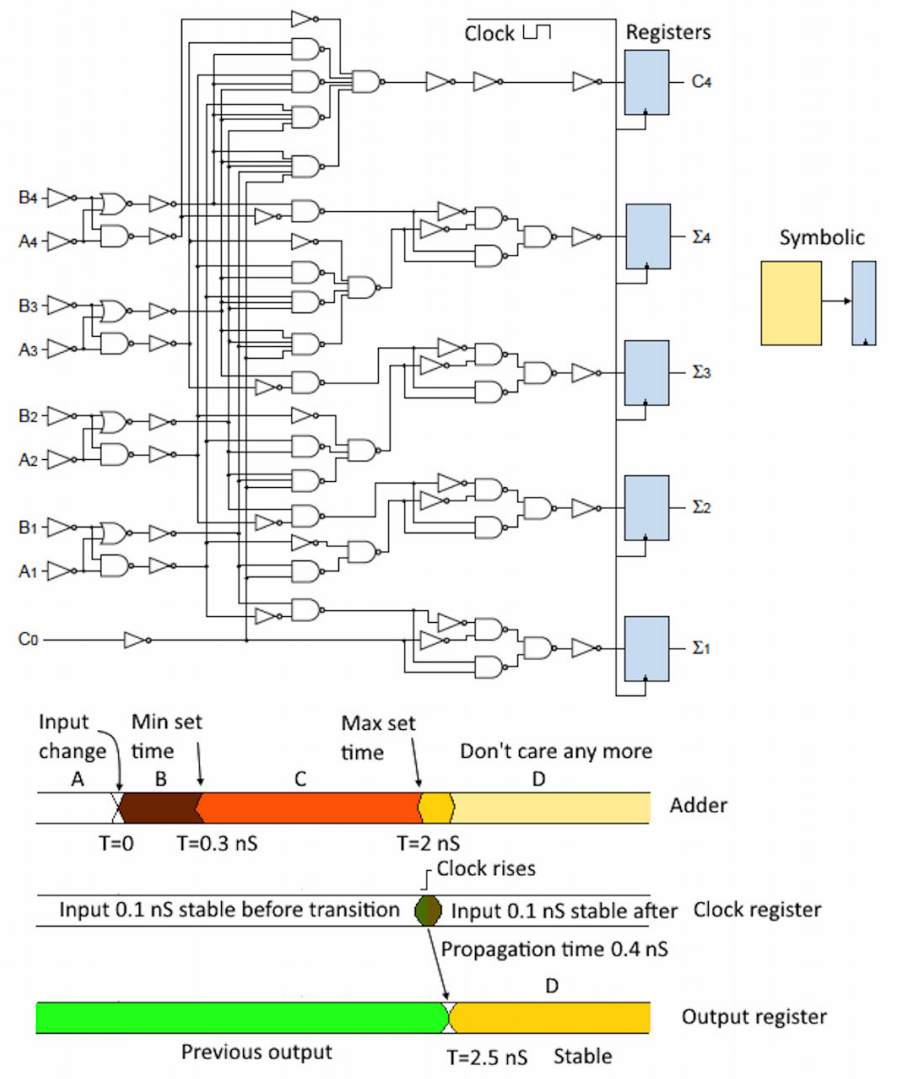
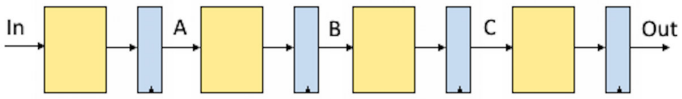


Figure 2.18: 4 bit adder with register
Source: Q-engineering

The output of the registers is updated at the rising edge of the clock signal. That is the only time the output can change. When the clock signal is high or low, the inputs cannot manipulate the output, then it remains stable. Just before the clock rises, the input must be stable for a minimum time, also just after the rise time. In contrast to the previous example, now consider a delay of 0.1 ns and adding the register's propagation time of (0.4 ns), the total propagation reach for a new stable signal is now 2.5 ns, which results increasing up to clock speed of 400 MHz. Observable in figure (2.19).



Clk	In	A	B	C	Out
⌐¬					
0		?	?	?	?
1			?	?	?
2				?	?
3					?
4					
5					
6					

Figure 2.19: Digital pipeline.
Source: Q-engineering

Taking into account the table in figure (2.19), each color represents a value. After four clock cycles, the value at the input has propagated through the network and appears at the output. Because the registers are updated simultaneously a new input is accepted every 2.5 ns. The propagation time has been restored. The time it takes to travel through the whole pipeline is called latency time and is 10 ns in the above diagram (4×2.5 ns). Every digital component is built on large pipelines which guarantee the required speed.[18]

2.7.4 The mul-add cell

The input signal of every neural node has effect on final result. This gives a simple multiplication for an input mediate by a weight value. The operation of multiplication can be implemented with same logic of an adder, then are necessary more gates. Remembering that the neural node has multiplied his value for weight value. Thus the representation assume the scheme in figure (2.20). Are necessary many important clarifications about the input registers `X1` and `X2` displayed in figure (2.21). They generate a delay line in the *mul-add* chain, thus can permit to correct synchronize. Considering the second cell *mul-add* who sum the output from previous cell (signal `A`) retarded by one clock cycle.

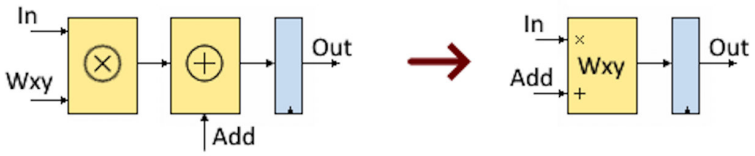


Figure 2.20: Mul-Add register.
Source: Q-engineering

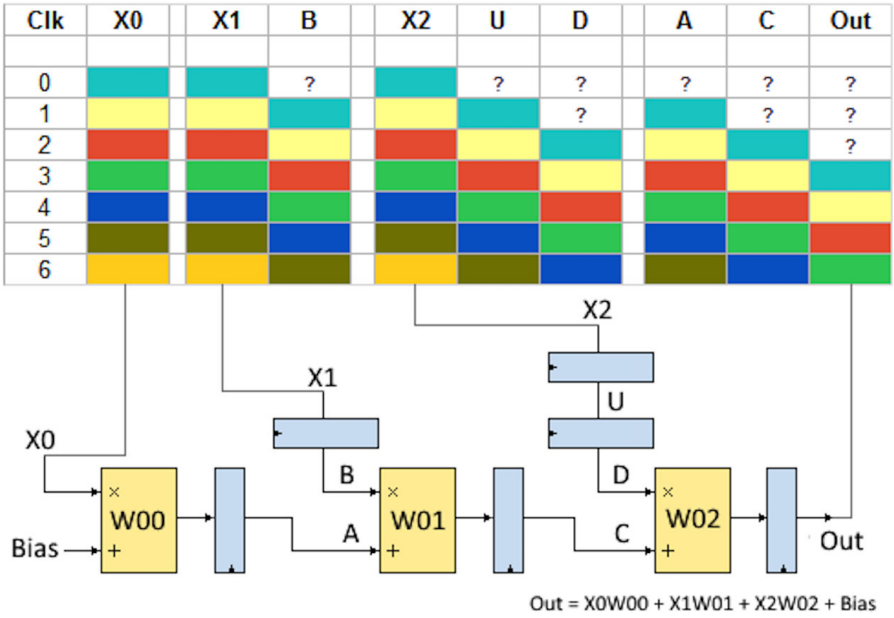


Figure 2.21: Schematic for a three input neuron.
Source: Q-engineering

Consequently also the multiplication between the value in register `X1` and weight value must be delayed the same amount of time. The schema reported in figure (2.21) shows for each colour that represent the vector of input $[X0, X1, X2]$ while the input `A` and `B`, represented by lines straight, maintains always same colour. This means that they appear simultaneous, i.e. are synchronised. The same must be valid for input `C` and `D`.

2.7.5 Systolic array

Starting from structure, just examined, is quite easy extend it to other neurons, taking into account that every single input is connected with all neurons in the next layer, as shows in figure (2.22). The adoption of this solution, called systolic array, permits to have an increment of speed in parallel calculation. Since the propagation time remains unchanged for an spread of systolic array in depth or width, while only the delay time increases the solution represented which is widely used for neural network hardware.

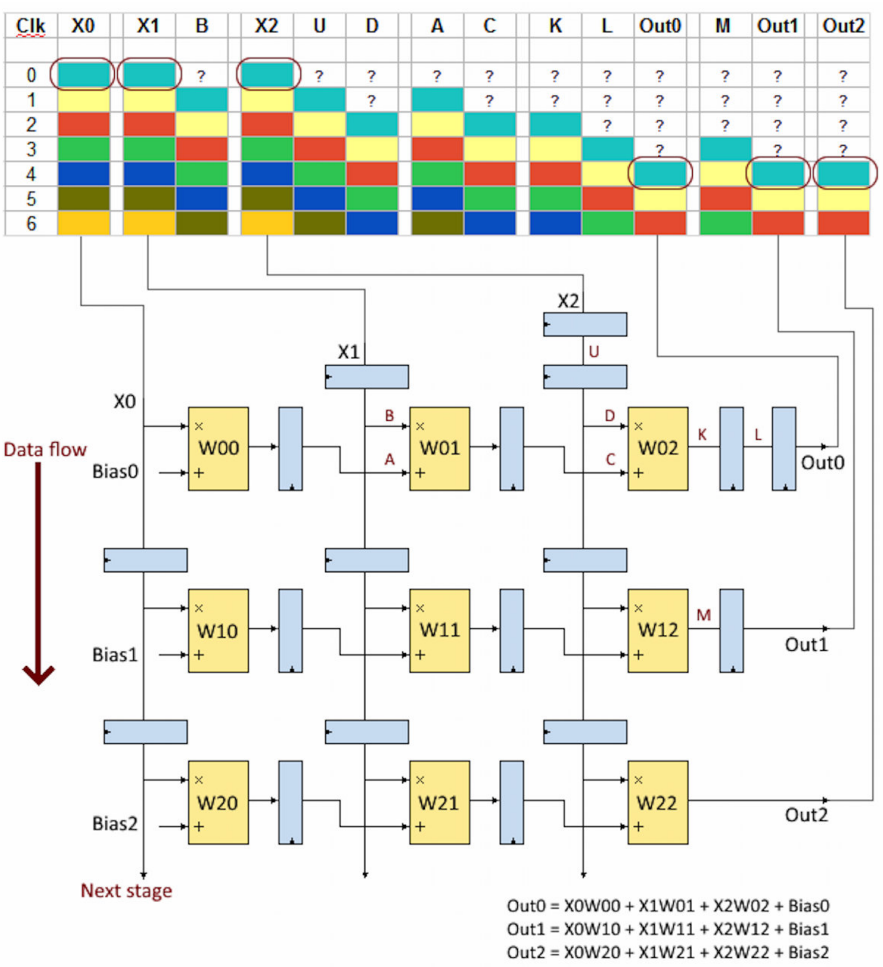


Figure 2.22: Systolic Array Edge TPU.
Source: Q-engineering

2.7.6 Activation unit

Once the output is available, it is sent to the activation unit. This module within the Edge TPU applies the activation function to the output. It is hardwired. In other words, you cannot alter the function, it works like a ROM. Probably it is a ReLU function as it is nowadays the most used activation function and it is very easy to implement in hardware.[18]

Chapter 3

Software

This chapter describes the software developed for the hardware described in the chapter 2. The choice to use the Qt framework guarantees the portability and cross-platform support, as well as the graphical interface and high data performance the feature of compiled languages. The software is divided into two programs capable of communicating with each other: the first works on Raspberry Pi and allows the user to interface easily with the FLIR thermal camera and with the Raspicam camera. It also acts as a TCP server to send the images shot to connected devices. The second part is a client program that receives the data sent by the main device, and allows the analysis of the image through machine learning on dedicated hardware.

3.1 Signals & Slots

In the project extensive use was made of the Signal and Slot mechanism, in order to allow communication between the objects in the code. Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. Signals and slots are made possible by Qt's meta-object system.[19]

3.1.1 Introduction

In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another. Other tool kits achieve this kind of communication using callbacks. A callback is a pointer to a function, so if you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function. The processing function then calls the callback when appropriate. While successful frame-

works using this method do exist, callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments.[19]

3.1.2 Signals and Slots

In Qt, we have an alternative to the callback technique: We use signals and slots. A signal is emitted when a particular event occurs. Qt's widgets have many predefined signals, but we can always subclass widgets to add our own signals to them. A slot is a function that is called in response to a particular signal. Qt's widgets have many pre-defined slots, but it is common practice to subclass widgets and add your own slots so that you can handle the signals that you are interested in.

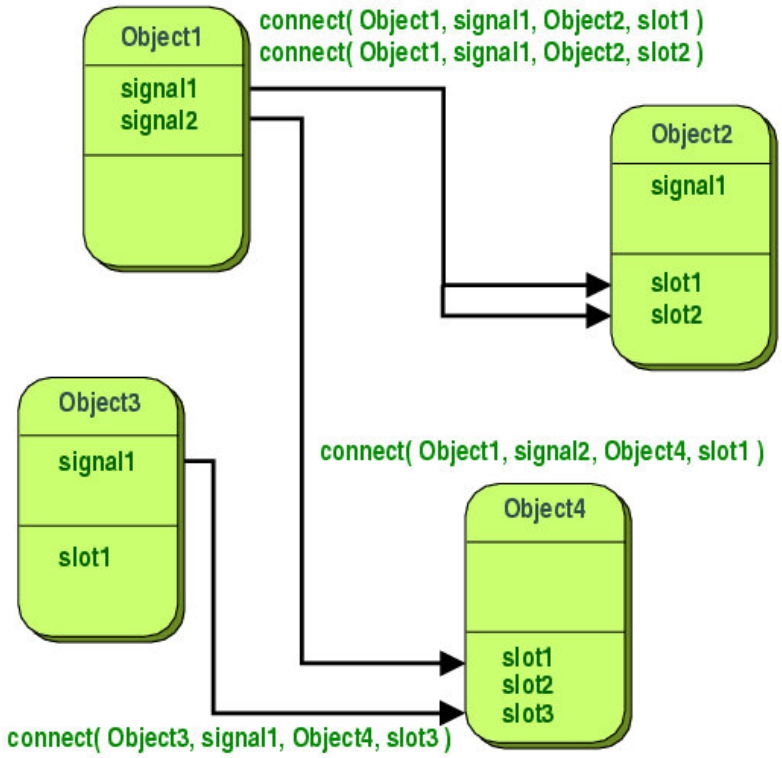


Figure 3.1: Signal and Slot scheme.
Source: https://doc.qt.io/qt-5/signalsandslots.html

The signals and slots mechanism is type safe: The signature of a signal must match the signature of the receiving slot. (In fact a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.) Since the signatures are compatible, the compiler can help us detect type mismatches when using the function pointer-based syntax. The string-based **SIGNAL** and **SLOT** syntax will detect type mismatches at runtime. Signals and slots are loosely coupled: A class which emits a signal neither

knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensures that if you connect a signal to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. They are completely type safe. All classes that inherit from `QObject` or one of its subclasses (e.g., `QWidget`) can contain signals and slots. Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation, and ensures that the object can be used as a software component. Slots can be used for receiving signals, but they are also normal member functions. Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be created with Qt. You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need. It is even possible to connect a signal directly to another signal. (This will emit the second signal immediately whenever the first is emitted.) Together, signals and slots make up a powerful component programming mechanism.[19]

3.1.3 Signals

Signals are emitted by an object when its internal state has changed in some way that might be interesting to the object's client or owner. Signals are public access functions and can be emitted from anywhere, but we recommend to only emit them from the class that defines the signal and its subclasses. When a signal is emitted, the slots connected to it are usually executed immediately, just like a normal function call. When this happens, the signals and slots mechanism is totally independent of any GUI event loop. Execution of the code following the emit statement will occur once all slots have returned. The situation is slightly different when using queued connections; in such a case, the code following the `emit` keyword will continue immediately, and the slots will be executed later. If several slots are connected to one signal, the slots will be executed one after the other, in the order they have been connected, when the signal is emitted. Signals are automatically generated by the *moc* and must not be implemented in the `.cpp` file. They can never have return types (i.e. use void).[19]

3.1.4 Slots

A slot is called when a signal connected to it is emitted. Slots are normal C++ functions and for this reason can be called normal; their only special feature is that signals can be connected to them. Since slots are normal member functions, they follow the normal C++

rules when called directly. However, as slots, they can be invoked by any component, regardless of its access level, via a signal-slot connection. This means that a signal emitted from an instance of an arbitrary class can cause a private slot to be invoked in an instance of an unrelated class. Compared to callbacks, signals and slots are slightly slower because of the increased flexibility they provide, although the difference for real applications is insignificant. In general, emitting a signal that is connected to some slots, is approximately ten times slower than calling the receivers directly, with non-virtual function calls. This is the overhead required to locate the connection object, to safely iterate over all connections (i.e. checking that subsequent receivers have not been destroyed during the emission), and to marshall any parameters in a generic fashion. While ten non-virtual function calls may sound like a lot, it's much less overhead than any `new` or `delete` operation, for example. As soon as you perform a string, vector or list operation that behind the scene requires `new` or `delete`, the signals and slots overhead is only responsible for a very small proportion of the complete function call costs. The same is true whenever you do a system call in a slot; or indirectly call more than ten functions. The simplicity and flexibility of the signals and slots mechanism is well worth the overhead, which your users won't even notice.[19]

3.2 Interface

As anticipated, the software run on the Raspberry Pi is made following the paradigm of object-oriented programming. Made in C++/Qt making extensive use of the proprietary classes of the framework and the Standard Template Library (STL). This program has some dependencies with regard to the thermal camera drivers supplied by the manufacturer and are 32-bit, since the Raspberry Pi 3, described in 2.1, is equipped with a 32-bit ARM Cortex-A53 processor. These allow total control of the camera. The second dependency is the Raspicam library which allows the interface with the RGB camera allowing the image acquisition. Continuing a user interface has been created through the use of widgets, this is divided into three areas. The first area shows the video streams acquired separately in two labels, the third larger label shows the two streams mixed through filters made available as default the overlay filter is used. It was preferred not to perform a match of the images with pixel-by-pixel recalculation for two reasons: the first due to the high difference in size of the sensors as that of the thermal camera is 80×60 pixels while the Raspicam is 3280×2464 pixels. Furthermore motivation is due to not aggravate the computational load on the CPU. The second area provides some controls for the user, in fact it is possible to save thermal images on files during the acquisition. you can change the heat map applied to the image on the fly by choosing from three different possibilities,

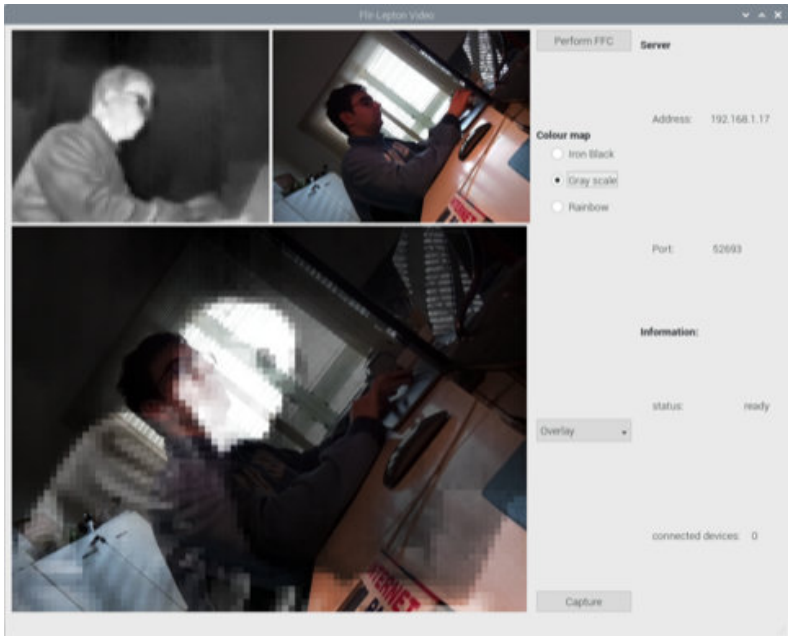
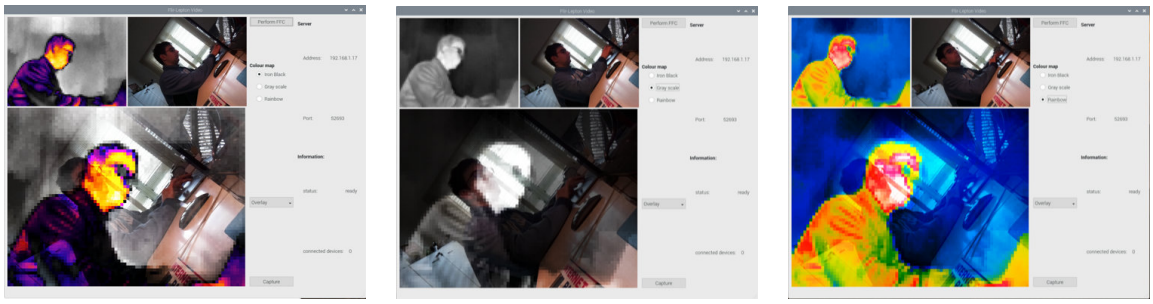


Figure 3.2: User interface run on Raspberry Pi 3b.

in the figure you can observe the result. Finally, it is possible to modify the mixing filter, also on the fly, of the video streams to obtain different effects to improve visibility or to increase details. The last section of the user interface shows the information relating to the TCP socket used for sending the video stream to other devices.



(a) *lava.* **(b)** *grayscale.* **(c)** *rainbow.*

Figure 3.3: Result colourization applied to thermal camera data.

3.2.1 Software Analysis

The user interface is run on the main thread, in order to maximize the performance of the executed code the image acquisition operation is performed in the secondary thread. Thanks to the use of the signal and slot system of Qt, introduced before in (3.1), it is possible to update the labels without blocks typical of multi-threaded programming. The difficulty of concurrent programming usually consists in synchronizing the access to

resources by different threads that act in competition on the same resources. Having two or more threads accessing the same data simultaneously can lead to unexpected and unwanted results. In fact, without the application of particular programming techniques, it is not possible to predict in a deterministic way, at the time of execution, when that specific thread will be executed: their progression depends on the priorities decided by the scheduler of the operating system and not by the programmer. In fact, multiple threads can access the same variable and modify its content or value. Therefore, synchronization techniques such as mutual exclusion are used to solve the problem. As a result, ideally a thread should execute code as independent of the rest of the program as possible. Furthermore, errors in synchronization between threads are often very difficult to detect because their occurrence essentially depends on the environment in which the program is run. The synchronization of one thread with another is normally necessary to allow them to communicate with each other and to return the results of a function to the main process; it is normally done through *mutex*[20]. Analysing the code of the thread that takes care of acquiring the images we can see that it proceeds without mutex[1], but uses the signal and slot system as mentioned before.

```
36 void LeptonThread::run() {
37   m_ir_image = QImage(80, 60, QImage::Format_RGB888);
38   leptonSPI_OpenPort(0);
39   usleep(LeptonLoadTime);
40   while (true) {
41     int resets = 0;
42     for (int j = 0; j < PACKETS_PER_FRAME; j++) {
43       read(spi_cs0_fd, result + sizeof(uint8_t) * PACKET_SIZE * j,
44            sizeof(uint8_t) * PACKET_SIZE);
45       int packetNumber = result[j * PACKET_SIZE + 1];
46       if (packetNumber != j) {
47         j = -1;
48         resets += 1;
49         usleep(LeptonResetTime);
50         if (resets == MaxResetsPerSegment) {
51           leptonSPI_ClosePort(0);
52           usleep(LeptonRebootTime);
53           leptonSPI_OpenPort(0);
54         }
55       }
56     }
57     m_frameBuffer = (uint16_t *)result;
58     int row, column;
59     uint16_t value;
60     uint16_t minValue = 65535;
61     uint16_t maxValue = 0;
62     for (int i = 0; i < FRAME_SIZE_UINT16; i++) {
63       if (i % PACKET_SIZE_UINT16 < 2) {
```

[1]The mutex class is a synchronization primitive that can be used to protect shared data from being simultaneously accessed by multiple threads.

```
64          continue;
65        }
66        auto temp = result[i * 2];
67        result[i * 2] = result[i * 2 + 1];
68        result[i * 2 + 1] = temp;
69        value = m_frameBuffer[i];
70        if (value > maxValue) {
71          maxValue = value;
72        }
73        if (value < minValue) {
74          minValue = value;
75        }
76        column = i % PACKET_SIZE_UINT16 - 2;
77        row = i / PACKET_SIZE_UINT16;
78      }
79      float diff = static_cast<float>(maxValue - minValue);
80      float scale = 255 / diff;
81      QRgb color;
82      for (int i = 0; i < FRAME_SIZE_UINT16; i++) {
83        if (i % PACKET_SIZE_UINT16 < 2) {
84          continue;
85        }
86        value = (m_frameBuffer[i] - minValue) * scale;
87        color = qRgb(this->colorMap[3 * value], this->colorMap[3 * value + 1],
88                     this->colorMap[3 * value + 2]);
89        column = (i % PACKET_SIZE_UINT16) - 2;
90        row = i / PACKET_SIZE_UINT16;
91        m_ir_image.setPixel(column, row, color);
92      }
93      QImage colour_image = cam->getImageRGB();
94      emit updateImage(m_ir_image);
95      emit updateCam(colour_image);
96      recalculateResult(m_ir_image.scaled(640, 480, Qt::KeepAspectRatio),
97                        colour_image.scaled(640, 480, Qt::KeepAspectRatio));
98    }
99    leptonSPI_ClosePort(0);
100 }
```

Listing 1: Infinite loop thread cameras.

Going to analyse in detail the infinite loop, reported in the listing (1), executed in a thread other than the main one that manages only the main interface, it can be observed that:

- (line 37–56) When the function starts, an instance of the `QImage` type object is instantiated to contain the image acquired during the cycle. The parameters of the frame size are provided and the color space this allows you to increase the speed of the cycle as it will not be cyclically cancelled and reallocated the same space, but will be reused. The communication with the thermal camera is opened via the SPI port if the cycle is not started or if the communication is interrupted the communication is closed.

- (line 66–78) The main cycle is to acquire data from the saved camera registers, as the order is MSB[2], reversed in according to LSB[3]. The values are then scaled to obtain a consistent representation of the hot and cold areas. Then the color map is applied to color the raw data obtained from the scaling process described above.

- (line 79–97) Finally, signals are output for the coloured thermal image, for the RGB image acquired by the Raspicam library and passed through the slot and the last resulting from mixing of previous two depend on effect of selected in UI interface.

As can be seen, the cycle proceeds without mutuals or blocking conditions, in fact, as previously described, it is possible to change the color map or the mixing tool on the fly.

[2]MSB can also stand for "most significant byte". *Big-endian processor*: When data is loaded into a multi-byte register, the first byte (with the lowest address) is the most significant byte of the data.[21]

[3]LSB can also stand for "least significant byte". *Little-endian processor*: When data is loaded into a multi-byte register, the first byte (with the lowest address) is the least significant byte of the data.[21]

3.3 Communication systems

In digital data communications, wiring together two or more devices is one of the first steps in establishing a network. As well as this hardware requirement, software must also be addressed. The Open System Interconnection (OSI) model proposed by the International Organization for Standardization (ISO) is a standard way to structure communication software that is applicable to any network. The model has been standardized by ISO and International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) which is the organization coordinating standards for telecommunications. The communication is based on low-level message passing between the communicating systems.

- A wants to communicate with B;
- A builds a message addressing B;
- A executes a call to the communication module to send the message to B.

Of course, A and B need to speak the same language, i.e., they need to agree on the meaning of the bits being sent. The protocols define the rules for communication. When data is exchanged through a computer network, the system rules are called a network protocol. A protocol must define the syntax, semantics, and timing of communication (i.e. how, what and when); the specified behaviour is typically independent on how it is to be implemented. Syntax: refers to the structure or the format of the data. Semantics: the way in which the bit patterns are interpreted. Timing: specify when the data can be sent and how fast it will be. Another term is synchronization. The communication between two nodes of a network occurs by sending messages. a message is broken down into a sequence of packets, and each packet is transmitted individually. The structure includes some control bits at the beginning of the message called *header* and at the end called *footer*. the control bits can contain information such as: the sending node of the packet, the recipient node of the package, package length information, information that allows you to verify the correctness of the package. [22]

3.3.1 Architecture

Client/server architectures are based on the functional division of IT applications into two categories. Few computers act as servers, run a particular program that allows the computer to receive requests and send replies. As shows in figure (3.4).
The clients, on the other hand, run a program that allows them to send requests and receive replies. Client/server applications are two-level architectures, that is, the first

level is the client and the second level is the server. The interaction protocols between client and server are quite simple. among the advantages of client/server architectures we have the simplicity of construction and the possibility of having easy-to-use clients. Disadvantages include the risk of overloading the computer acting as a server and the communication channel with which the server is connected to the network.[22]

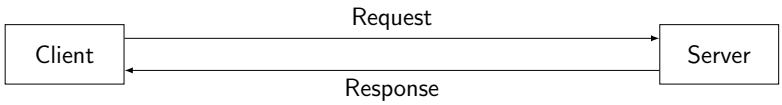
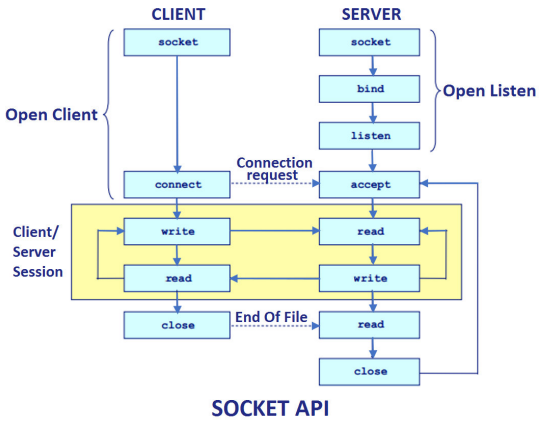
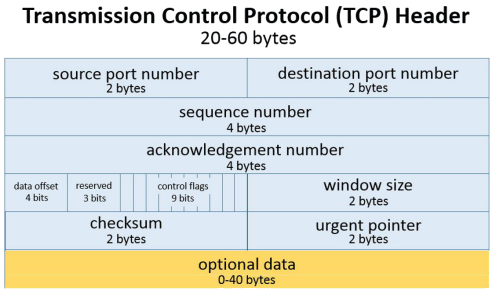


Figure 3.4: Two level client/server architecture

In the Internet protocol suite there is also the Transmission Control Protocol (TCP). It is the complement of the Internet Protocol, yielding the well known TCP/IP. TCP provides reliable, ordered, and error-checked delivery of the messages between applications on an IP network. The most used internet applications, i.e. the World Wide Web, e-mail, file transfer, rely on TCP.



(a) *server socket through which the server can listen to client sockets and establish a connection*



(b) *TCP Header Format*

Figure 3.5: TCP protocol.
Source: Lifewire

3.3.2 Server implementation

In our case, it implements `QTcpSocket` network communication. Therefore, there is a server service in main program shows in figure (3.6). While there is a client application see section (3.4). Server application has `QTcpSocket` and it listens to some port, in our case 52693. Client has `QTcpSocket`, but has not connected to the server yet:

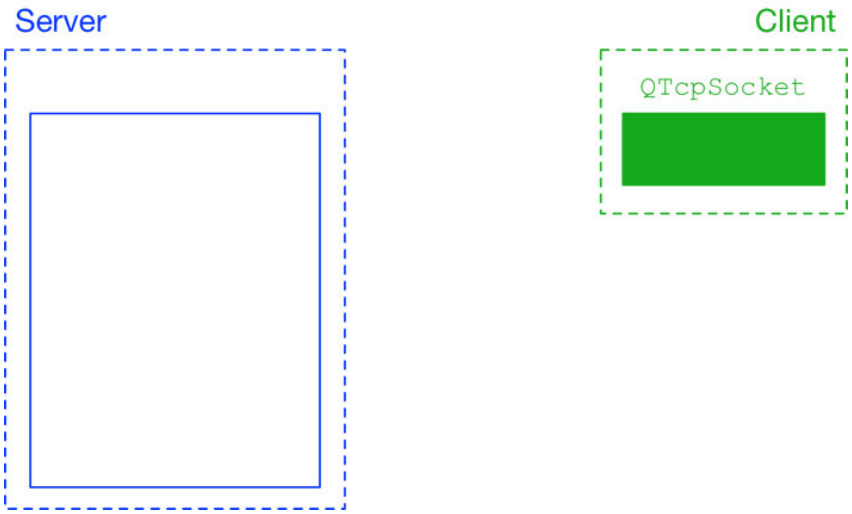


Figure 3.6: Interface client server `QTcpSocket`.

When client connects to server, a `QTcpSocket` is created on the server's side, through which server and client can talk to each other and start send message, shows in figure (3.7). In particular, we observe the function reported in (2) implemented in server ap-

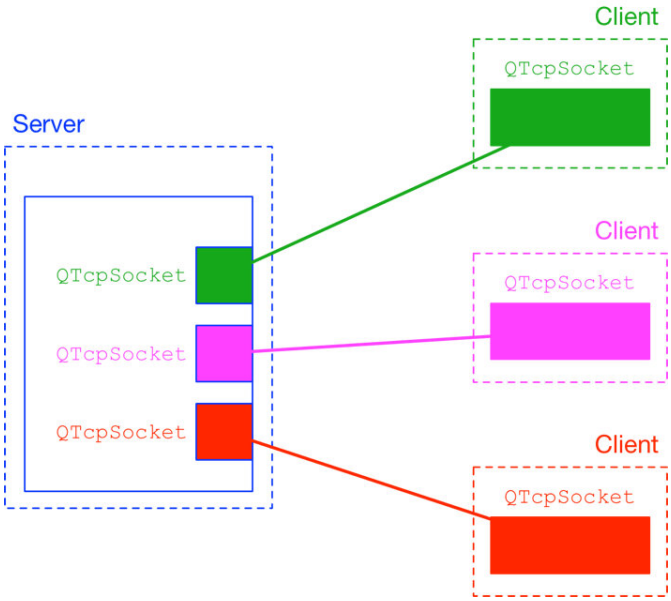


Figure 3.7: Connection between server-clients.

plication which sends the image just captured by the RGB camera to the server to be encapsulated in the message before sending. In the function it is possible to observe the presence of a mutex which is locked to protect the resource and avoid overwriting through function calls. The frame, i.e. the resource, is converted and stored in buffer

before sending.
Upon exiting the function body, the mutex is unlocked, thus giving free access to resources again.

```
connect(this, &MainWindow::update_rgb_image, [=](QImage image) {
  QMutexLocker locker(&mutex);
  QPixmap img =
      QPixmap::fromImage(image.scaled(512, 512, Qt::KeepAspectRatio));
  QByteArray bImage;
  QBuffer bBuffer(&bImage);
  // Putting every image in the buffer
  bBuffer.open(QIODevice::ReadWrite);
  img.save(&bBuffer, "JPG");
  // Sending to TCPServer function to display the image
  server->is_newImg(bImage);
});
```

Listing 2: Particular report function sending image.

3.4 Client

The software that acts as a client on the Coral dev-board, presented in chapter (2.6), created by Google is based on the same Qt framework previously introduced in order to guarantee portability and reliability of the code. This, remember, has an ARM Cortex-A53 processor, but unlike the one mounted on Raspberry it executes 64-bit code and takes advantage of the new **armv8a** architecture with a significant performance gain. This difference arises from the execution of Machine Learning operations supported by the TPU, described in detail in section (2.7). As seen for the main program, here too we find a main interface executed in the main thread. The simplest interface is divided into two areas: the first one where it is possible to observe the flow of images coming from the TCP socket. As hows in figure (3.7).
On the other hand, the second zone offers the possibility of connecting to the TCP socket by entering the address and port of the machine on which the server is run, which is listening for possible connection requests. Once the connection between the server and the client is established, the label is updated with the images received.

3.4.1 Software Analysis

To avoid blockages and unpleasant delays in receiving images from the TCP socket, multi-thread programming was used. In fact, as previously described, the interface managed by QWidget is run on the main thread. If the connection is stable, the thread starts which allows reception in a queue waiting to leave.

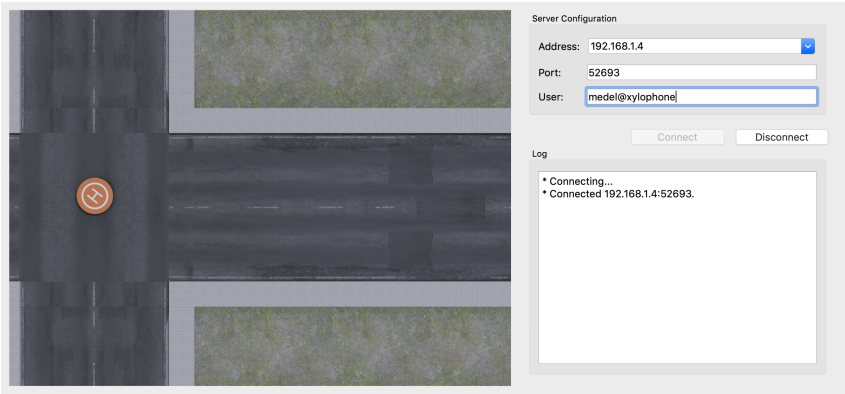


Figure 3.8: User interface client application.

On the other hand, this is prepared when the class is instantiated within the main function. Since this infinite cycle is the critical factor, we analyse its structure in detail below.

```
12  void StreamerThread::run() {
13    QMutexLocker locker(&mutex);
14    socket = new QTcpSocket;
15    socket->connectToHost(m_address, m_port);
16    QByteArray buffer;
17    while (m_quit == false) {
18      if (socket->waitForReadyRead(3000)) {
19        buffer.append(socket->readAll());
20        msleep(350);
21        emit newImageAvailabe(buffer);
22        buffer.clear();
23      }
24    }
25    socket = nullptr;
26  }
```

Listing 3: Particular report function sending image.

As you can see in the function code, shown in Listing (3), we observe the instance of the `socket` member object of the `QTcpSocket` class type. By starting the connection, the server starts to transmit the frame. The critical section from the *while loop* is protected by a mutex, highlighted by the `QMutexLocker` class, a mutex indicates a process of synchronization between concurrent processes or threads, with which multiple parallel tasks are prevented from simultaneously accessing data in memory or other resources subject to race condition.[23]
Locking and unlocking a `QMutex` in complex functions and statements or in exception handling code is error-prone. `QMutexLocker` can be used in such situations to ensure that the state of the mutex is always well-defined. `QMutexLocker` should be created within a function where a `QMutex` needs to be locked. The mutex is locked when `QMutexLocker` is created. If locked, the mutex will be unlocked when the `QMutexLocker` is destroyed.

Using `QMutexLocker` greatly simplifies the code, and makes it more readable.[24]

The buffer is filled with reading from the socket and before putting the signal to update the image in the interface a small interval of time is waited to guarantee the complete reception of the image. Before updating the label on the dashboard, filter the image to verify that it is consistent and different from an empty or corrupt image, shown in listing (4). If this occurs, the function is immediately exited to avoid viewing an image being ready to receive a new image.

```
88  void TcpClient::imageAvailabe(QByteArray baImage) {
89    QPixmap pixImage;
90    QImage image;
91
92    if (!pixImage.loadFromData(baImage, "JPG")) return;
93    image = pixImage.toImage();
94    if (image.pixel(image.width() - 1, image.height() - 1) == 4286611584 &&
95        image.pixel(image.width() / 2, image.height() - 1) == 4286611584 &&
96        image.pixel(0, image.height() - 1) == 4286611584)
97      return;
98
99    emit updatePixmap(pixImage);
100 }
```

Listing 4: Implementation filter for empty JPEG image.

3.5 Inference

In this section the inference process is analyzed on the neural network model, taking into account that the two applications share the same code. In fact, when the model is loaded into the buffer, it is interpreted by the TensorFlow Lite library, returning the training information of the model and in particular the information regarding the input and output of the model. For this case, where you are interested in the problem of object detection, the size of the image is requested at the input and the number of color channels in this case is constant and fixed at three. Furthermore, the function reported in listing (5) is analyzed, which allows you to adapt the model input with the data, that is, the images coming from the camera, taking into account the nedd to satisfy the neural model input and to adapt the information for the tensor calculation if in floating point, or in 8-bit integers. It also takes into account the process of quantization that took place during the transformation from original model to lite model, in fact the input will also be recalculated. For fully integer models, the inputs are unsigned integer 8-bit.
The mean and standard deviation (`std_dev`), calculated as in (3.1), values specify how to `uint8` values map to the float input values used while training the model.

```
template <class T>
void formatImageTFLite(T* out, const uint8_t* in, int image_height,int image_width, int image_channels,
    int wanted_height,int wanted_width, int wanted_channels,bool input_floating) {
  constexpr float input_mean = 127.5f;
  constexpr float input_std = 127.5f;
  int number_of_pixels = image_height * image_width * image_channels;
  std::unique_ptr<tflite::Interpreter> interpreter(new tflite::Interpreter);
  int base_index = 0;
  // two inputs: input and new_sizes
  interpreter->AddTensors(2, &base_index);
  // one output
  interpreter->AddTensors(1, &base_index);
  // set input and output tensors
  interpreter->SetInputs({0, 1});
  interpreter->SetOutputs({2});
  // set parameters of tensors
  TfLiteQuantizationParams quant;
  interpreter->SetTensorParametersReadWrite(0, kTfLiteFloat32, "input",{1, image_height, image_width,
    image_channels}, quant);
  interpreter->SetTensorParametersReadWrite(1, kTfLiteInt32, "new_size", {2},quant);
  interpreter->SetTensorParametersReadWrite(2, kTfLiteFloat32, "output",{1, wanted_height, wanted_width,
    wanted_channels}, quant);
  tflite::ops::builtin::BuiltinOpResolver resolver;
  const TfLiteRegistration* resize_op =resolver.FindOp(tflite::BuiltinOperator_RESIZE_BILINEAR, 1);
  auto* params =
    reinterpret_cast<TfLiteResizeBilinearParams*>(malloc(sizeof(TfLiteResizeBilinearParams)));
  params->align_corners = false;
  interpreter->AddNodeWithParameters({0, 1}, {2}, nullptr, 0, params, resize_op,nullptr);
  interpreter->AllocateTensors();
  // fill input image
  // in[] are integers, cannot do memcpy() directly
  auto input = interpreter->typed_tensor<float>(0);
  for (int i = 0; i < number_of_pixels; i++) input[i] = in[i];
  // fill new_sizes
  interpreter->typed_tensor<int>(1)[0] = wanted_height;
  interpreter->typed_tensor<int>(1)[1] = wanted_width;
  interpreter->Invoke();
  auto output = interpreter->typed_tensor<float>(2);
  auto output_number_of_pixels =wanted_height * wanted_height * wanted_channels;
  for (int i = 0; i < output_number_of_pixels; i++) {
    if (input_floating)
      out[i] = (output[i] - input_mean) / input_std;
    else
      out[i] = (uint8_t)output[i];
  }
}
```

Listing 5: function that adapts the input image to the input required by the neural model.

The mean is the integer value from 0 to 255 that maps to floating point `0.0f`.

$$\texttt{std\_dev} = \frac{255}{\texttt{float}_{\max} - \texttt{float}_{\min}} \tag{3.1}$$

Different hardware may have preferences and restrictions that may cause slight deviations when implementing the specification that result in implementations that are not bit-exact. Whereas that may be acceptable in most cases, the nature of machine learning (and deep learning in the most common case) makes it impossible to provide any hard guarantees.[25]

8-bit quantization approximates floating point values using the following formula (3.2).

$$\texttt{real}_{\texttt{value}} = (\texttt{int8}_{\text{value}} - \texttt{zero}_{\text{point}}) \times \texttt{scale} \tag{3.2}$$

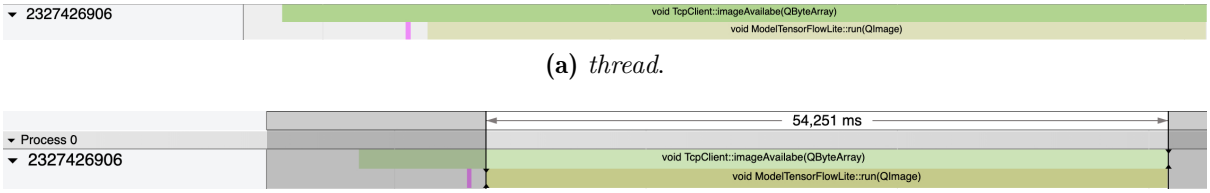
Per-axis (aka per-channel in Conv ops) or per-tensor weights are represented by `int8` two's complement values in the range $[-127, 127]$ with zero-point equal to 0. Per-tensor activations/inputs are represented by `int8` two's complement values in the range $[-128, 127]$, with a zero-point in range $[-128, 127]$.

3.5.1 Architecture x86_64

We analyze in particular the inference cycle following the reception of the frame by the TCP socket described previously in section (3.4.1). In figure (3.9a) the captured process shows the functions performed:

- The dark green rectangle represents the socket's reception function.
- The lighter green rectangle, on the other hand, is the time needed to calculate the inference.
- The pink rectangle represents the time to update the image in the user interface.

Although only one thread is shown here, in the solution adopted the TensorFLow Lite module uses 8 cores, but it is possible to set a number of cores greater than or equal to 1. Comparing the two figures (3.9c) and (3.9b) it is observed that the duration of the cycle is approximately 64.376 ms and the other hand the infercence time is 54.251 ms. This time includes the execution of the functions described above, from this it is possible to deduce that the most onerous task in terms of time and the reconstruction of the image by the socket inside the **TcpClient::imageAvailable** function. Meanwhile, **ModelTesnorFLowLite::run**'s function starts after about 10 ms to end almost simultaneously with the function. Then the cycle repeats itself. Performing the inference with the TensorFlow Lite model, described in section (3.5), on a laptop equipped with an Intel i7-7820HQ CPU 2.90 GHz processor shows the benefits due to compression and quantization of the neural model. In particular, it is observed that the average time is 78 ms therefore a considerable performance boost if compared with the time required, when using the uncompressed model, is considerably slower when compared to the result obtained in this case. Although it is important to note that if the required time drops on the one hand, the number of false positive cases increases on the other.



(a) *thread.*

(b) *time inference.*

(c) *total time cycle.*

Figure 3.9: Visual benchmark inference on `x86_64` architecture.

3.5.2 Architecture armv7l

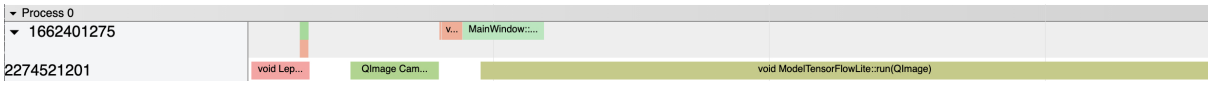
We now analyze the result of the execution of the inference process on the hardware of the Raspberry Pi 3b computer board which mounts an ARM cortex-A53 processor. In fact, the figures (3.10) show that the processes present a different number and row, this is because parallel architecture is exploited and in particular in the benchmark it has been possible to capture the distribution of the tasks on the threads. Unlike what is seen in the previous section, the inference is made immediately without considering the expectation of the streaming of images by the socket. The work cycle examined has a total duration of 694.296 ms, please note that here too more than one core is assigned to the inference process. Moving forward in the analysis, we can distinguish within the cycle:

- In green the image acquisition by the camera in the second thread.
- The orange rectangle the updating of the interface with the new image in the first thread.
- In bright green the execution of the function in the first thread that passes the image acquired by the camera to the function that performs the inference.
- The segment filled in olive green rectangle the inference process performed in the second thread.

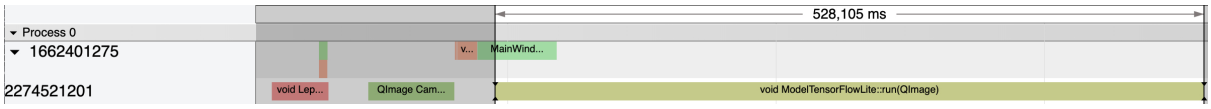
As can be seen in figure (3.10b), the inference process by the **ModelTesnorFLowLite::run** function has an execution time of 528.105 ms, considerably slower than the `x86_64` architecture previously discussed. although it has a longer execution time, the whole program is not affected by this aspect due to the optimization carried out by the compiler and by exploiting the multi-threading architecture as well as the possibility of using SIMD[4] instructions.

[4]Single Instruction Multiple Data

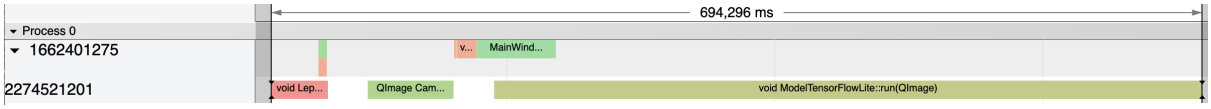
Finally, we want to remember that both data come from the same code executed by the machine, but two different architectures are being compared. The first encountered `x86_64` supports 64-bit instructions, benefiting from the larger size of the registers and in greater numbers, despite a much higher clock frequency and being able to count on 4 physical and 4 virtual cores. On the other hand the ARM process with `armv7l` architecture executes 32 bit instructions and has a much lower clock frequency.



(a) *thread.*



(b) *time inference.*



(c) *total time cycle.*

Figure 3.10: Visual benchmark inference on `armv7l` architecture.

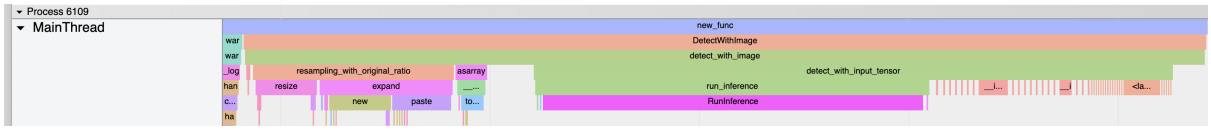
3.5.3 Architecture armv8a and TPU

The last comparison is made on the processing capacity of Google's Coral Dev-Board and in particular by the TPU that makes available for the tensor calculation. In the visual benchmark created in figure (3.11a), unlike the previous ones, there is a single thread. In this case it was not possible to isolate only the functions of interest, but all the calls to the functions made by the program were extracted. In this analysis, unlike the previous ones, you are not using all the threads of the processor that you remember is a 64-bit ARM cortex-A53.

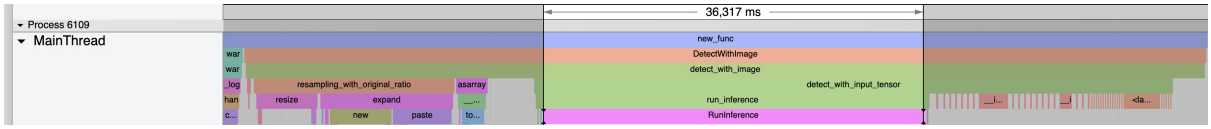
The calculation is moved to the TPU thus relieving the main processor from the calculation of the inference on the quantized model and compiled specifically to operate on the unit. Improvement can be observed with respect to the case of the execution of the inference on the `armv7l` architecture in that when the times drop drastically, in fact they are equal or better than to those seen in section (3.5.1).

The TPU processing unit performs the inference in approximately 36.317 ms highlighted by the pink segment reported in figure (3.11b) so called the **RunInference** function.

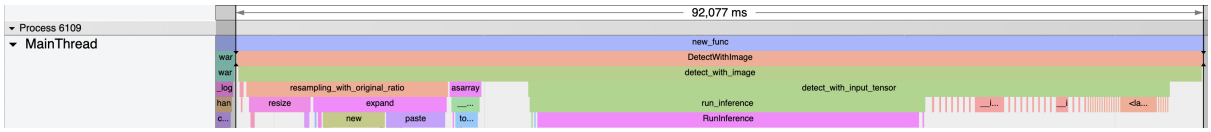
As seen in (2.7), the advantage of using TPU to perform largely derives from the adoption of 8-bit integers that make tensor calculation easier.



(a) *thread.*



(b) *time inference.*



(c) *total time cycle.*

Figure 3.11: Visual benchmark inference on `TPU`.

Furthermore, the structure of the systolic array also guarantees benefits due to the possibility of keeping the calculation time constant as the tensor increases. The entire inference calculation cycle is 92.077 ms in which the image arrives. The image is scaled to the desired input of the tensor and ready to be analyzed, as observable in figure (3.11c).

Chapter 4

Neural Networks

Deep Learning has exploded to the present day because it has become accessible to the general public. Often its contribution is noticeable in intelligent systems as assistants and interpreters of natural language and prototypes of self-driving vehicle systems. In this document, we try to provide a thorough investigation of deep learning in its applications and mechanisms. In particular, as a categorical state of the art collection in research on deep learning. An impetus in deep learning technologies was imparted by Google in 2015 thanks to TensorFlow. TensorFlow supports computation on multiple CPUs and GPUs, with optional CUDA and SYCL extensions. In addition, TensorFlow Lite is designed for mobile and embedded machine learning, and provides anAndroid Neural Networks API.[26]

4.1 Introduction

In this era, a large amount of structured and unstructured data has become available. **Machine Learning** (**ML**) has evolved as a branch of artificial intelligence: it envisages the development of self-learning algorithms, which are capable of acquiring knowledge from data with the aim of making predictions. Instead of requiring a human presence who manually enacts the rules and builds models for the analysis of large amounts of data, machine learning offers a more efficient alternative to capture the knowledge in the data. Machine learning aims to gradually improve the performance of forecasting models and to make data driven decisions. In this section we will examine the three different types of machine learning: *supervised learning*, *unsupervised learning* and *reinforcement learning*. Where we will show the fundamental differences between these types of learning.[27]

4.1.1 Supervised learning

The main purpose of supervised learning is to derive a model from training data, which allows us to make predictions for data that are not available or future. Here, the term "supervision" refers to the fact that the output signal labels of the sample sets are already known. A supervised learning task, which is based on discrete class labels, is also called a classification task, in figure (4.1) it is possible to observe a process diagram. Another supervised learning subcategory is regression, whose resulting signal is a continuous value. Classification is a sub-category of supervised learning, which has the goal to provide class category labels for new instances, based on observations made in the past. These labels are discrete, unordered values that can be considered as belonging to a group of instances. However, the set of class labels does not necessarily have to be a binary nature.
The predictive model identified by a supervised learning algorithm can consider each class label that is present in the learning dataset of a new instance, which is not labelled. A typical example of *multi-class classification* is the recognition of hand-written text.[27]

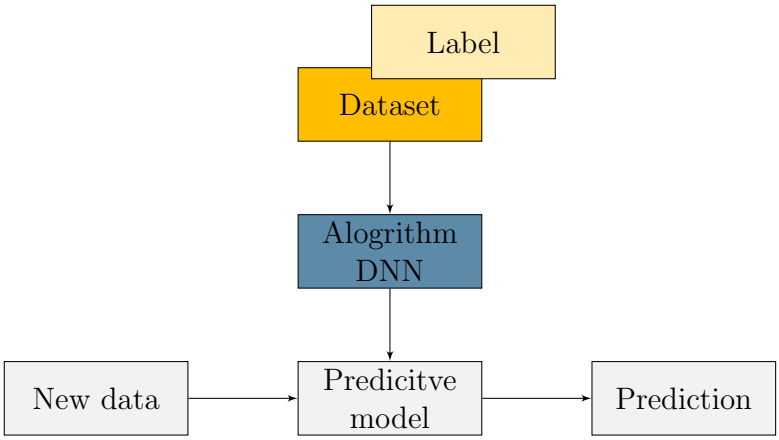


Figure 4.1: supervised learning scheme

4.1.2 Reinforcement learnig

Another type of machine learning is reinforcement learning. Here, the goal is to develop a system (*agent*) for people to improve their performance. In order to do so, that system is based on interactions with the environment. Since information relating to the current state of the environment also include a *reward* signal, we can consider strengthening learning as an example of supervised learning. However, this feedback is not the correct label or the true value of truth, but it represents the quality of the measurement of the performance measured by the reward function. Through interaction with the environment, an agent can then use reinforcement learning to learn a series of actions, which maximize this reward through a trial-and-error exploratory approach or deliberative planning.[27]

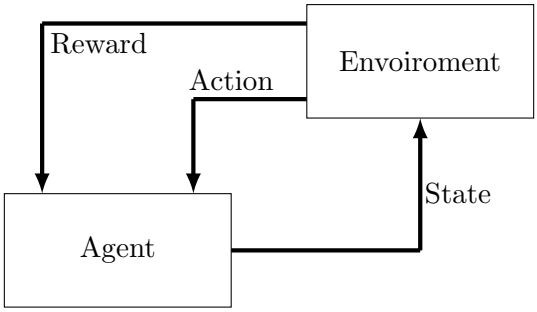


Figure 4.2: reinforcement learning scheme

4.1.3 Unsupervised learning

In supervised learning, we know in advance the correct answer when we describe our model, while in reinforcement learning we define a measure, or reward, for the specific actions performed by the agent. In unsupervised learning, on the other hand, we are dealing with unlabelled data or data from the unknown structure. Using unsupervised learning techniques, we are able to observe the structure of our data, to extract meaningful information from them without being able to rely on the guide of a variable known relative result, or a reward function. Clustering is an exploratory technique of data analysis that allows us to organize a series of information within meaningful groups (*cluster*) without having any previous knowledge of memberships in such groups. Each cluster that can be derived during the analysis defines a group of objects that share a certain degree of similarity, but which are more dissimilar than the objects present in the other clusters, which is why clustering is sometimes called *"unsupervised classification"*. Clustering is an excellent technique for structuring information to identify meaningful relationships in the data.[27]

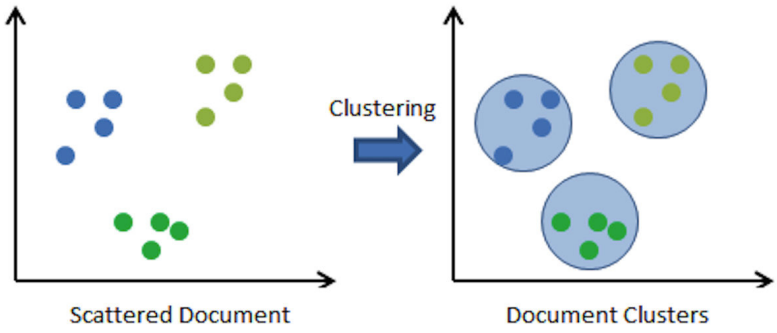


Figure 4.3: example of clustering

4.2 Deep Learning for Object Detection

Object detection is fundamental computer vision problem visual recognition other than image classification, semantic segmentation. In particular object detection not only recognizes object categories but also predicts the location of each object by a bounding box. On the other hand semantic segmentation aims to predict pixel-wise classifiers to assign a specific category label to each pixel, thus providing an even richer understanding of an image. However, in contrast to object detections, semantic segmentation does not distinguish between multiple objects of the same category.[28]
Current state-of-the-art object detection systems are variants of the following approach: hypothesize bounding boxes, re-sample pixels or features for each box, and apply a high quality classifier. Although accurate, these approaches have been too computationally intensive for embedded systems and, even with high-end hardware, too slow for real-time or near real-time applications. Often detection speed for these approaches is measured in seconds per frame, and even the fastest high-accuracy detector, the basic Faster R-CNN, operates at only 7 frames per second (FPS). There have been a wide range of attempts to build faster detectors by attacking each stage of the detection pipeline, but so far, significantly increased speed comes only at the cost of significantly decreased detection accuracy.[29] To derive a performance improvement translated into an increase in detection speed with high precision (58 FPS with mAP 72.1% on VOC2007 test, vs Faster R-CNN 7 FPS with mAP 73.2% or YOLO 45 FPS with mAP 63.4%). From this derives the elimination of the proposals of delimitation boxes and of the subsequent phase of refilling of the pixels or of the characteristics. Furthermore, the improvements introduced can be summarized in the use of a convolution filter to predict the categories of objects and offsets in the positions in the positioning of the panes, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple features maps from the later stages of a network in order to perform detection at multiple scales. It is observed that with these modifications we can achieve high-accuracy detection using relatively low resolution input and further increasing processing speed. While these contributions may seem independently small, we note that the resulting system improves accuracy on high-speed detection for PASCAL VOC from 63.4% mAP for YOLO to 72.1% mAP for the network used.[29, 30] So as to ensure stability and portability of the model The API TensorFlow Object Detection is used, an open source framework based on TensorFlow which simplifies the construction, training and implementation of object detection Templates.[31]

Figure 4.4: example multiple object detection in a single image.
Source: GitHub.com - TensorFlow

4.2.1 Single Shot Detector MobileNet

The model of SSD is based on the idea that we want to detect the position of an object of interest, in addition to knowing which one is classified. To better comprehend let's start from nomenclature:

Single Shot The localization and classification operation must be a single forward pass of the network.

Multibox Technique for bounding box regression.

Detector The neural network has the duty classifies those detected object.

The model *SSD MobileNet* exploit four stages: the input layer for importing the target image, the MobileNet base for extracting image features, the SSD for classification regression and bounded box regression and the output layer for exporting the detection result.[32] The advantage of this net are speed and accuracy in detection task borrowed from reduced compute complexity.

[1]AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1.

Model name	Speed (ms)	COCO mAP[1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco	26	18	Boxes
ssd_mobilenet_v1_quantized_coco	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco	29	16	Boxes
ssd_mobilenet_v1_ppn_coco	26	20	Boxes
ssd_mobilenet_v1_fpn_coco	56	32	Boxes
ssd_resnet_50_fpn_coco	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes

Table 4.1: Performance comparison model based on SSD MobileNet.
Source: GitHub - TensorFlow model zoo

4.3 Datatset

Particular attention is needed in the construction of a good training dataset, in fact, as seen before in (4.1.1), we deal with a supervised learning where we know the response of our labels and bounding box positioned objects. A script is provided that can build a dataset divided into folders: training, validation and testing; as you can see in figure (4.5). A large number of figures per sample that clearly highlight the characteristics that you want to study allows a greater rate of success of the training preventing the *overfitting*. Acquiring a large number of images is not always achievable, using some augmentation techniques, which virtually allow to increase the observability of images, for example, by rotating, distorting and translating them.
The two train and validated folders are essential for the addition of the neural network. Instead, the test folder contains a set of images that the network has never seen and so it is necessary to measure the degree of confidence acquired in the network.

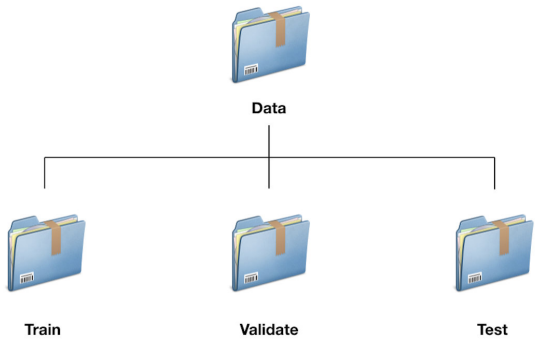


Figure 4.5: Dataset structure

4.3.1 Landing zone dataset

The dataset was artificially constructed with Blender a 3D graphics program.(4.6)
The object of interest are the drone landing mats. These have different shapes and colors in fact they are available in various color shapes. The signs on these also vary from the classic H to X to more or less conspicuous symbols. Thus three models were created, two with a circular plan and one with a square plan. Textures were then applied to these two models to obtain a faithful representation of that of concrete objects.

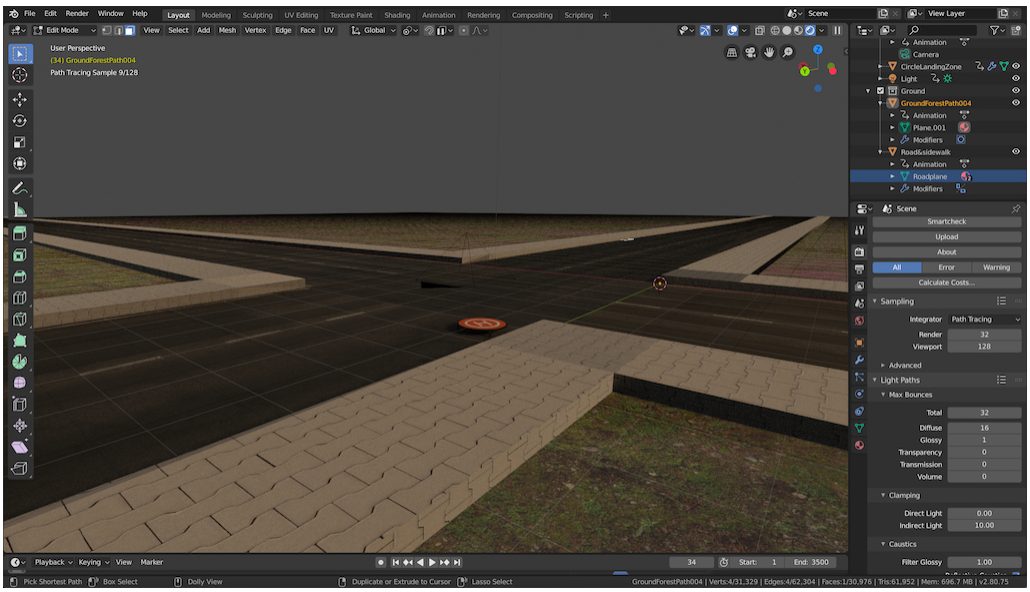


Figure 4.6: Modelling phase of the scenario in Blender.

After the construction of the carpet models, it was necessary to contextualise them in credible scenarios. Thus, three main scenarios were created: first scenario placed in the middle of a road junction. Second scenario near a straight road flanked by a side-walk. Third zone is set in the countryside. To take the shots, some photographic factors were taken into account, in fact they are generated starting from the technical characteristics of the Raspberry camera presented in section (2.2).
Moreover, thanks to a simple script, the trigger points were generated with respect to the landing mat position, these positions vary in height, width and depth with respect to the carpet placed on the ground. A scheme is visible in figures (4.7).

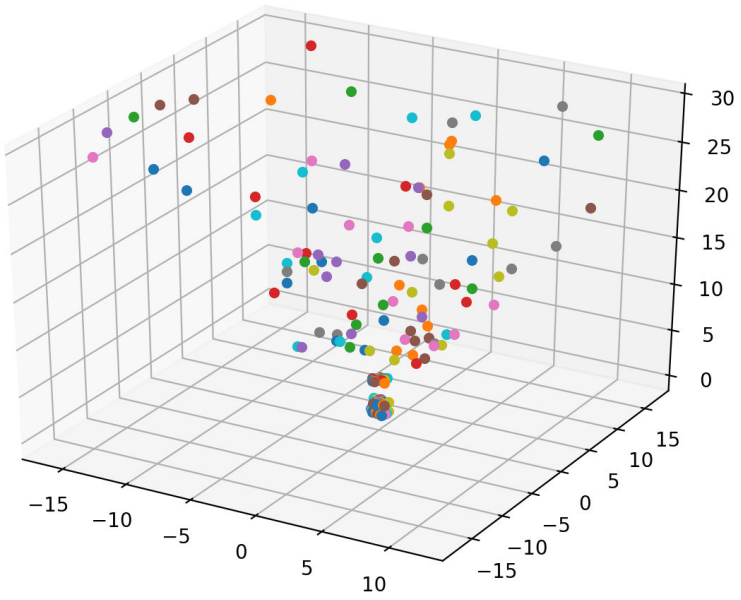


Figure 4.7: Configuration camera position.

To increase the variety of rendered shots, a very basic day-night cycle was used to detect the color variation. All the shots are taken with a top view as seen in the examples shown in (4.9). After completing the shots they annotated themselves highlighting the position of the object of interest in the various shots and positions.
The process is carried out with the aid of a software that extracts and generates the annotations. As shown in figure (4.8).

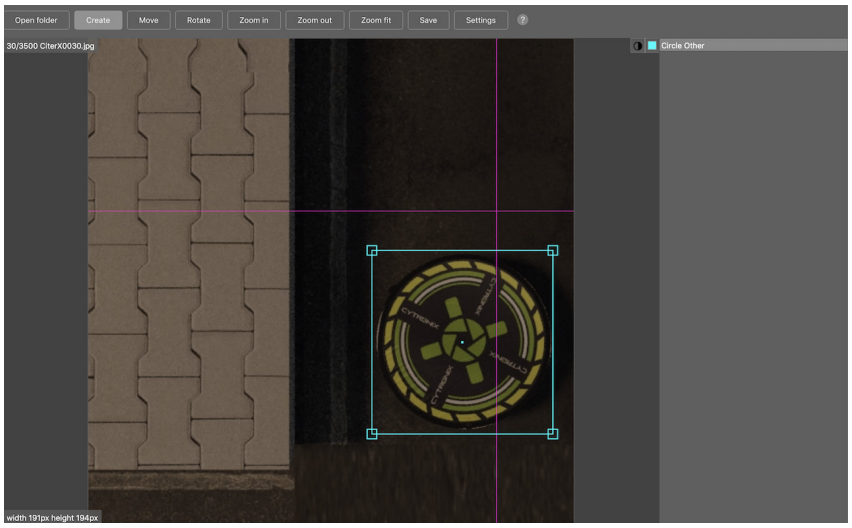


Figure 4.8: Process of annotation.

4.3.2 Thermal imaging dataset

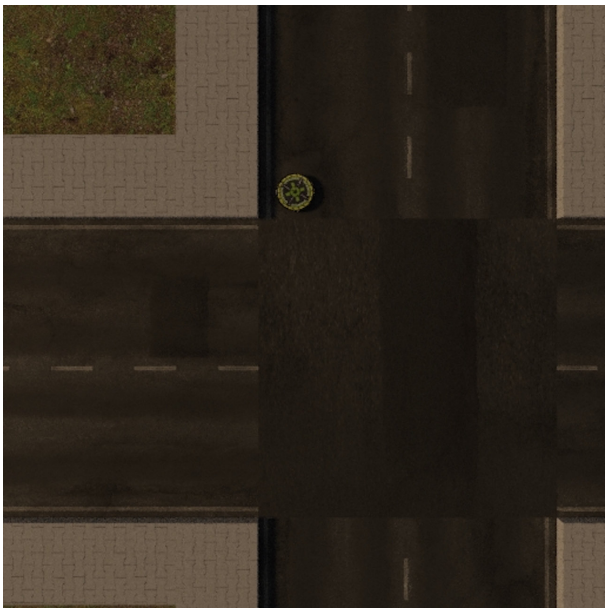
The dataset created and distributed free of charge by FLIR[2] was used to train the neural network on the recognition and classification of objects in thermal images. The ability to sense thermal infra-red radiation, or heat, within the ADAS context provides both complementary and distinct advantages to existing sensor technologies such as visible cameras, LiDAR and radar systems: With over 15 years of experience in automotive, FLIR has the only automotive-qualified thermal sensor that is deployed in over 500,000 cars today for driver warning systems. The FLIR thermal sensors can detect and classify pedestrians, bicyclists, animals and vehicles in challenging conditions including total darkness, fog, smoke, inclement weather and glare, providing a supplemental dataset beyond LiDAR, radar and visible cameras. The detection range is four times farther than typical headlights. When combined with visible light data and distance scanning data from LiDAR and radar, thermal data paired with machine learning creates a more comprehensive detection and classification system.[33]

The dataset provided by FLIR was created from videos collected from a moving vehicle in Santa Barbara, California, USA covering roads and highways in different weather conditions. Although there are many objects in the images to be catalogued, only ten types of objects have been selected, in particular:

- Category 1: People
- Category 2: Bicycles - bicycles and motorcycles (not consistent with coco)
- Category 3: Cars - personal vehicles and some small commercial vehicles.
- Category 17: Dogs

The boxes around the objects of interest are as narrow as possible in fact: when occlusion occurred, only non-occluded parts of the object were annotated. Heads and shoulders were favoured for inclusion in the bounding box over other parts of the body for people and dogs. When occlusion allowed only parts of limbs or other minor parts of an object to be visible, they were not annotated. Wheels were the important part of the Bicycles category. Bicycle parts typically occluded by riders, such as handlebars, were not included in the bounding box. People riding the bicycle were annotated separately from the bicycle. When an object was split by an occlusion, two separate annotations were given to the two visible parts of the object.

[2]https://www.flir.com/oem/adas/adas-dataset-form/



(a) *middle of a road junction (sunrise).*

(b) *countryside.*

(c) *close position at the crossroads.*

Figure 4.9: Examples shots scenarios after render

(a) *person.*

(b) *cars and people.*

(c) *bicycle and persons.*

Figure 4.10: Thermal image extracted from FLIR dataset.
Source: https://www.flir.com

4.4 Training and Result

The training process is execute at the university's HPC at the University of Trento, with the aim of minimizing also the topics of this operation which is particularly expensive in terms of calculation resources and time. For this reason, it was decided to optimize the training procedure by modifying only some of the parameters of interest, i.e. applying fine tuning techniques to preserve the weights of a previous training and modifying only the input layer and expand the final layer by adding new classification categories. The two most important parameters are:

- Input image resolution: the corresponding value can be $128, 160, 192, 224$ pixels or greater. In this project, a size of 512 pixels was chosen, this in order not to penalize excessively the input images. At high altitudes, as explained for the realization of the dataset in section (4.3.1), the details and targets becoming extremely difficult to detect therefore a high scaling worsens the result. Meanwhile this choice negatively affects the duration of the entire training by lengthening the processing times.

- The second parameter relating to the model: this value allows you to establish the size of the model starting from the complete one, i.e. it is possible to train a fraction of the model, choosing from the values: $1, 0.75, 0.50$ or 0.25. The higher the fraction chosen, the greater the precision of the model.

```
25 export PIPELINE_CONFIG_PATH=$MODEL_DIR/pipeline.config
26 export TF_RESEARCH_MODEL_DIR=$PROJECT_DIR/tf-models/research
27
28 export NUM_TRAIN_STEPS=50000
29 export SAMPLE_1_OF_N_EVAL_EXAMPLES=1
30
31 # From the project/tf-models/research/ directory
32 # Make sure you've updated PYTHONPATH
33 python3 ${TF_API_DIR}/object_detection/model_main.py \
34     --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
35     --model_dir=${MODEL_DIR} \
36     --num_train_steps=${NUM_TRAIN_STEPS} \
37     --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
38     --alsologtostderr
```

Listing 6: Train script setup.

The above commands (6) are used to set the selected model, the folder of the dataset containing the images, the labels of the target objects and the destination folder of the trained model. The `num_train_steps` parameter set to 50000 indicates how many times training iterates over data. The higher this value, the more accurate the result will be.

4.4.1 Result

In the graphs shown in the figures (4.11, 4.12) it is possible to observe the progress of the training process on the quantities of interest. In particular we observe that:

- `rpn_class_loss` RPN anchor classifier loss;
- `rpn_bbox_loss` RPN bounding box loss graph;
- `class_loss` loss for the classifier head of CNN;
- `bbox_loss` loss for bounding box refinement;

Each of these loss metrics is the sum of all the loss values calculated individually for each of the regions of interest. The general loss metric given in the log is the sum of the other five losses (it possible check it by summing them up). The values of the classification losses depend on the confidence value associated with the real class. Consequently, these values reflect the confidence of the model in predicting and labelling a class, that is, verifying how correct the prediction is.
The bounding box loss values reflect the distance between the true box parameters that is, the (x, y) coordinates of the box location, its width and its height and the predicted ones. This value derives from the loss of regression and penalizes major absolute differences. In particular, it will exhibit a behaviour for small differences, on the other hand a linear compaction for large differences. Ultimately this aspect reveals to us how correct the localization of the objects within the image is, that is, this value allows us to evaluate how precisely the model foresees the area that contains the object. This information is extracted by TensorBoard is another great debugging and visualization tool. The model is configured to log losses and save weights at the end of every epoch.[34]

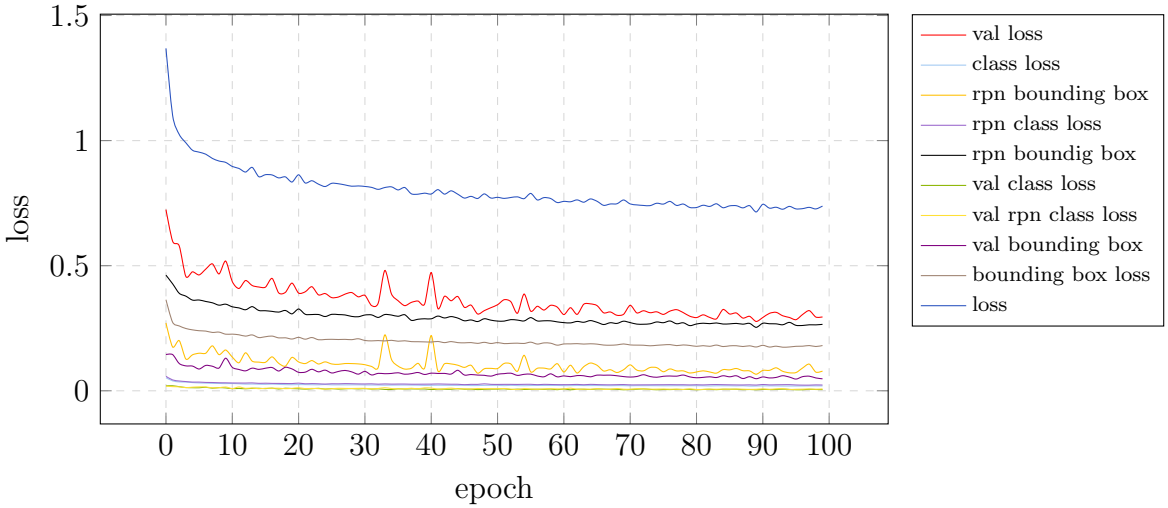


Figure 4.11: Result training object detection on last ouptut layer.

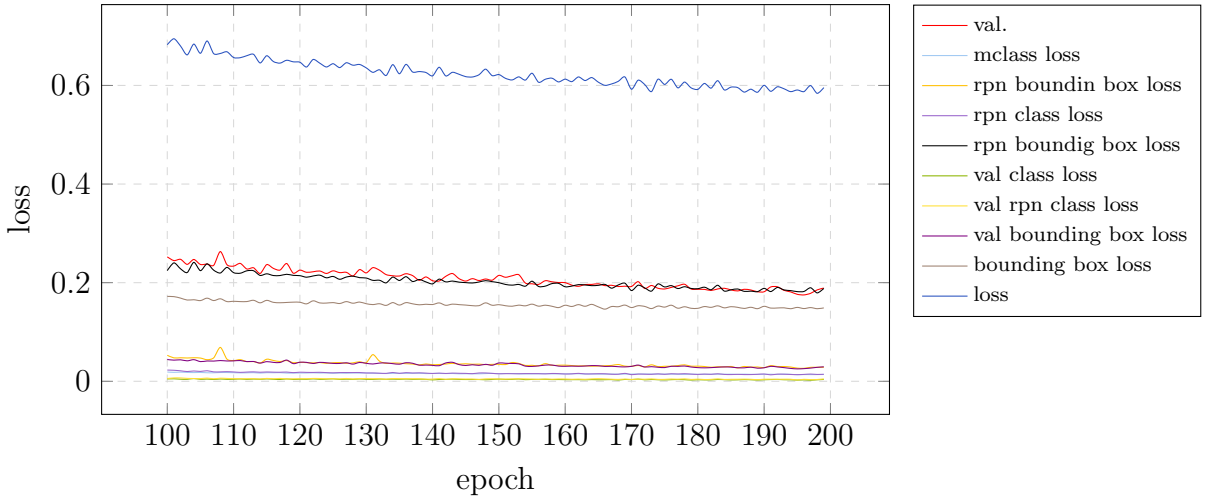
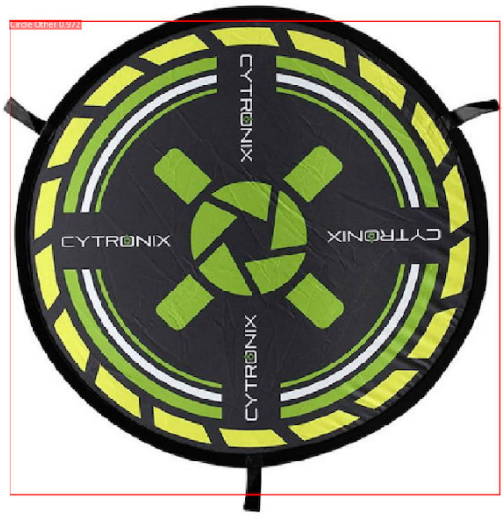


Figure 4.12: Result training classification.

Finally, some examples of inference are given to evaluate the response of the trained model as can be seen in figures (4.13). Although images taken from the internet have been provided and taken from real environments using some demonstrations that those generated by the computer; the model is able to provide the correct answer even when the target object is covered with elements.



(a) *product image*

(b) *real environment - grass*

(c) *real environment - sand*

(d) *render from dataset*

Figure 4.13: Evaluation of the model response with images.

4.5 Quantizzation

Lastly, after evaluating the reliability of the model, it is necessary to quantify and convert it to run on mobile devices, embedded devices or other specific architectures such as ARM processors available on Raspberry and mobile phones, as well as on TPU using the Google dev Board discussed in section (2.6). TensorFlow Lite is TensorFlow's lightweight solution for mobile and embedded devices. It lets you run machine-learned models on mobile devices with low latency, so you can take advantage of them to do classification, regression or anything else you might want without necessarily incurring a round trip to a server. It's presently supported on Android and iOS via a C++ API. The interpreter can also use the API for hardware acceleration, otherwise it will default to the CPU for execution. TensorFlow Lite is comprised of a runtime on which you can run pre-existing models, and a suite of tools that you can use to prepare your models for use on mobile and embedded devices.

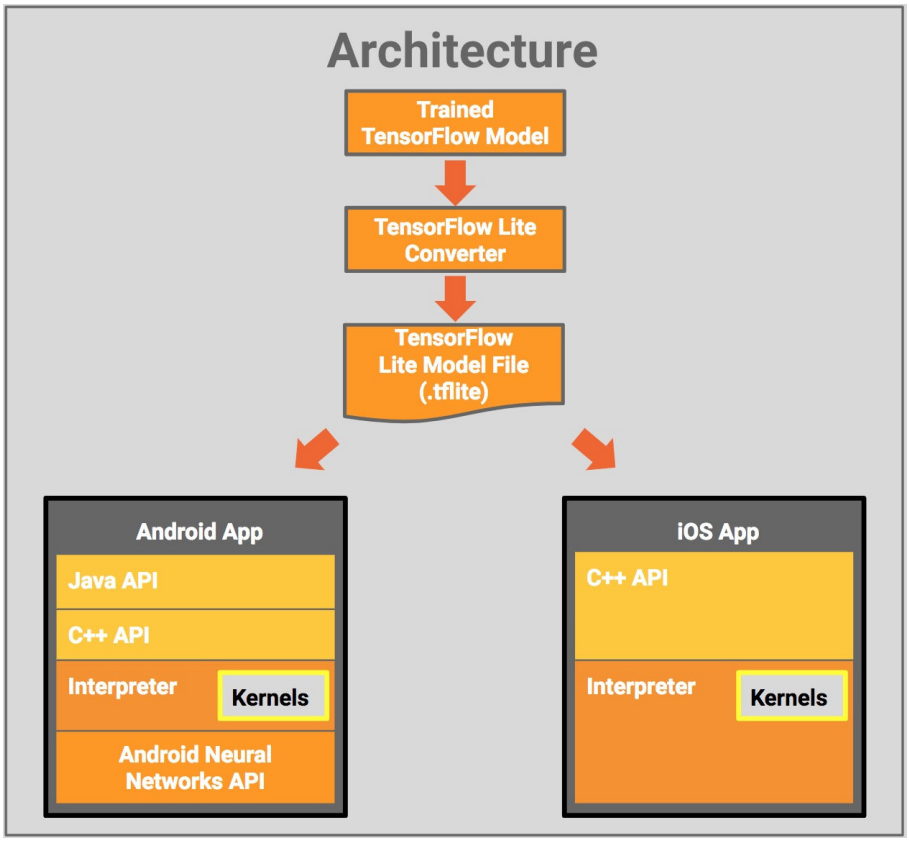


Figure 4.14: TensorFlow Lite Architecture.
Source: [35]

The tools made available by TensorFLow Lite allow you to convert a model trained on computing devices such as HPC clusters, the conversion operation supported by a set of core operators, both quantized and float, which have been tuned for mobile platforms.

The core operations incorporated pre-fused activations and biases to further enhance performance and quantized accuracy. TensorFlow Lite supports using custom operations in models. From this operation a more compact and better optimized model for portable applications is obtained. This allows you to increase the speed of execution of the model on devices such as Coral dev-Board, but not only.
The `.tflite` model is obtained from these commands which will perform the quantization.

```
16 ###############################################################################
17 # Convert to TensorFlow Lite
18 ###############################################################################
19 # Run the export_tflite_ssd_graph.py script to get the frozen graph
20 python3 ${TF_API_DIR}/object_detection/export_tflite_ssd_graph.py \
21     --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
22     --trained_checkpoint_prefix=${CHECKPOINT_PATH} \
23     --output_directory=${OUTPUT_DIR} \
24     --add_postprocessing_op=true \
25     --inference_type=QUANTIZED_UINT8 \
26     --mean_values=128 \
27     --std_values=128 \
28     --change_concat_input_ranges=false \
29     --allow_custom_ops
30
31 cd tf-models/tensorflow
32 bazel run -c opt tensorflow/lite/toco:toco -- \
33     --input_file=${OUTPUT_DIR}/tflite_graph.pb \
34     --output_file=${OUTPUT_DIR}/detect.tflite \
35     --input_shapes=1,512,512,3 \
36     --input_arrays=normalized_input_image_tensor \
37     --output_arrays='TFLite_Detection_PostProcess',\
38         'TFLite_Detection_PostProcess:1',\
39         'TFLite_Detection_PostProcess:2',\
40         'TFLite_Detection_PostProcess:3' \
41     --inference_type=QUANTIZED_UINT8 \
42     --mean_values=128 \
43     --std_values=128 \
44     --change_concat_input_ranges=false \
45     --allow_custom_ops \
46     --default_ranges_min=0 --default_ranges_max=6
```

Listing 7: Train script setup.

The biggest advantage is obtained when the evaluation of the neural network takes place as they involve numerous calculations that drastically affect the energy consumption of the processor and consequently on the battery used to power the device. The simplification of the model allows you to use less memory and cycles as it is possible to switch from 32-bit floating point calculations to calculations with 8-bit integers. This has repercussions on processors that have high performance and reduced consumption.

Chapter 5

Conclusion

The aim of this thesis is to increase automation and calculation skills on board drone with the help of the machine learning, solving the computer vision problem and in particular object detection. The system uses a standard camera with a resolution of 8 MegaPixels to acquire images that will be processed by the neural network, as it can rely on the Lepton 2.5 thermal camera to detect objects that they emit heat. In particular, the software created is designed to exploit as much as possible the parallelization of the processor, precisely to manage the flow videos from the cameras. The Raspberry Pi 3b has wifi connection therefore the software uses it on a TCP socket to send two flows to a possible device connected to the same network. Thus it is possible to carry out the inference with the model directly on board neural that in the case of color images will try to identify the carpet landing, while in the case of the thermal camera the second model neural will try to identify any obstacles, in this case people, to avoid them during the execution of the maneuvers. As discussed the use of deep learning allows you to solve a specific problem calibrating the response on the input data to have the classification in response of the object and its position within the frame captured by the camera. In particular using the single shot detection model based on MobileNet, it provides excellent results both in the classification of subjects framed as well as the position of bounding box that frames the same. The fine tuning techniques thus allow you to train the in a short time model to classify a specific target. Clearly this is not possible to do on board the drone, but high devices are required computational services such as the University HPC cluster made available. Using the TensorFlow Lite framework to compress the model once it has been trained, so you can choose to use floating point or integer values. In particular, using a compressed model allows it to be used on embedded devices, typically less performing, as they can be computer board card, such as the Raspberry Pi 3b used in this project, thus providing acceptable performance. The TPU mounted on the Coral Dev-Board was used instead of the processors shows how it is still possible to increase the

response speed of the neural model thanks to the heavy use of 8-bit integers. Although the benefits are significant the compression reached penalizes the final result at the end of the inference process due to the big difference between the architectures of the processors, the optimizations introduced by the compiler and the user affect the recognition of the target thus producing a slight increase in cases of false positives. Although it is possible to code any model in TensorFlow Lite this is it is not always possible, in fact by using custom layers they can prevent the compression of the trained model. For this reason, during the development phase of the neural network, it was decided to use a network from the TensorFlow API unlike the preliminary phases in which a customized model had been adopted, therefore the model is more compatible with TensorFlow Lite. It presents the right balance between accuracy and speed of execution, this also ensured full compatibility with the TPU. Ultimately it is possible to remark that the work done here produces a system able to increase the capabilities of a drone thanks to the help of deep learning without resorting to pre-existing solutions deriving from closed licenses or commercial combined with devices for sale.

Chapter 6

Future work

GIVENthe results achieved, further improvements to the software are possible to the entire project structure. It will be possible to modify the neural model to make it more efficient and precise by using new backbone networks that perform the classification with the possibility of extracting new ones features from the pictures. The dataset can be reviewed, expanded or calibrated for more specific objectives, I provide images rich in detail and if needed I can provide possible images made with dedicated hardware. This project has focused attention on implementation and training of a neural network model for embededd systems, therefore on an architecture that would give the best results. Although a 3D dataset has been used for area shooting, in the future for a dataset made with images captured in real life flight conditions, this would lead to more detail and realism on the perception of the environment. Then by improving the data provided to the process training it will be possible to obtain a more accurate response of the model. In fact, note that for the inference on the thermal images a dataset was made by FLIR with a different hardware from the one used in this project. Making a new dataset with the use of the Lepton 2.5 camera will allow to capture images with a different resolution which allow you to capture different characteristics for the benefit of the results of the image analysis both in the training phase and in response to the computer vision problem. If on the one hand the results of the neural network are connected to the dataset to train them, on the other hand you can intervene by changing the hyper-parameters of the model. By modifying the hyper-parameters it is possible to carry out more selective calibrations during training by changing the balance of accuracy and speed. Although the compression offered by TensorFLow Lite is an excellent tool to work on embedded devices, the quantization shows a drift of the results with respect to the uncompressed model. Starting from the version of TensorFLow 2.0 it is possible to train the network with quantized values. This will allow you to obtain better results during the execution of the inference and a drastic reduction of false positives.

Bibliography

[1] P. Boucher, "Domesticating the drone: The demilitarisation of unmanned aircraft for civil markets," *Science and Engineering Ethics*, vol. 21, no. 6, pp. 1393–1412, 2015. [Online]. Available: https://doi.org/10.1007/s11948-014-9603-3

[2] P. Daponte, L. De Vito, F. Lamonaca, F. Picariello, S. Rapuano, and M. Riccio, "Measurement science and education in the drone times," in *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, 2017, pp. 1–6.

[3] X. Wang, "Autonomous mobile robot visual SLAM based on improved CNN method," *IOP Conference Series: Materials Science and Engineering*, vol. 466, p. 012114, dec 2018. [Online]. Available: https://doi.org/10.1088%2F1757-899x%2F466%2F1%2F012114

[4] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," 2016.

[5] S. Y. Choi and D. Cha, "Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art," *Advanced Robotics*, pp. 1–13, 03 2019.

[6] S. Budiansky, *Air Power: The Men, Machines, and Ideas That Revolutionized War, from Kitty Hawk to Iraq*. Penguin Publishing Group, 2005. [Online]. Available: https://books.google.it/books?id=AQrsfvhieMgC

[7] Wikipedia contributors, "Unmanned aerial vehicle — Wikipedia, the free encyclopedia," 2020, [Online; accessed 25-febbraio-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unmanned_aerial_vehicle&oldid=941712049

[8] E. Upton, "Raspberry pi 2 on sale now at $35," *Raspberry Pi*, 2016.

[9] M. A. Pagnutti, R. E. Ryan, G. J. C. V, M. J. Gold, R. Harlan, E. Leggett, and J. F. Pagnutti, "Laying the foundation to use Raspberry Pi 3 V2 camera module imagery

for scientific and engineering purposes," *Journal of Electronic Imaging*, vol. 26, no. 1, pp. 1 – 13, 2017. [Online]. Available: https://doi.org/10.1117/1.JEI.26.1.013014

[10] R. Foundation. (2016) New 8-megapixel camera board on sale at $25. [Online]. Available: https://www.raspberrypi.org/blog/new-8-megapixel-camera-board-sale-25/

[11] ——. (2016) Camera module. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera/

[12] M. Vollmer and K.-P. Möllmann, "Teaching physics and understanding infrared thermal imaging ," in *14th Conference on Education and Training in Optics and Photonics: ETOP 2017*, X. Liu and X.-C. Zhang, Eds., vol. 10452, International Society for Optics and Photonics. SPIE, 2017, pp. 474 – 484. [Online]. Available: https://doi.org/10.1117/12.2266142

[13] D. Cardone and A. Merla, "New frontiers for applications of thermal infrared imaging devices: Computational psychopshysiology in the neurosciences," *Sensors*, vol. 17, 05 2017.

[14] A. Rogalski, *Infrared Detectors*, ser. Electrocomponent science monographs. Taylor & Francis, 2000. [Online]. Available: https://books.google.it/books?id=4b3WLgomvd0C

[15] M. Vollmer and K. Möllmann, *Infrared Thermal Imaging: Fundamentals, Research and Applications*. Wiley, 2017. [Online]. Available: https://books.google.it/books?id=0-Q3DwAAQBAJ

[16] *FLIR LEPTON® Long Wave Infrared (LWIR) Datasheet*, 1st ed., October 2014.

[17] Flir lepton breakout board v1.4 by getlab | groupgets. [Online]. Available: https://groupgets.com/manufacturers/getlab/products/flir-lepton-breakout-board-v1-4

[18] Google coral edge tpu explained in depth - q-engineering. [Online]. Available: https://qengineering.eu/google-corals-tpu-explained.html

[19] Signals & slots | qt core 5.14.1. [Online]. Available: https://doc.qt.io/qt-5/signalsandslots.html

[20] Wikipedia, "Thread (informatica) — wikipedia, l'enciclopedia libera," 2019, [Online; in data 11-febbraio-2020]. [Online]. Available: http://it.wikipedia.org/w/index.php?title=Thread_(informatica)&oldid=107249669

[21] D. V. James, "Multiplexed buses: the endian wars continue," *IEEE Micro*, vol. 10, no. 3, pp. 9–21, June 1990.

[22] D. Mandrioli, *Informatica: arte e mestiere*, ser. Collana di istruzione scientifica. McGraw-Hill Companies, 2008. [Online]. Available: https://books.google.it/books?id=aae9NwAACAAJ

[23] Wikipedia, "Mutex — wikipedia, l'enciclopedia libera," 2020, [Online; in data 11-febbraio-2020]. [Online]. Available: http://it.wikipedia.org/w/index.php?title=Mutex&oldid=110301420

[24] Qmutexlocker class | qt core 5.14.1. [Online]. Available: https://doc.qt.io/qt-5/qmutexlocker.html#details

[25] Tensorflow lite 8-bit quantization specification. [Online]. Available: https://www.tensorflow.org/lite/performance/quantization_spec?

[26] W. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. PP, pp. 1–1, 04 2018.

[27] S. Raschka, *Machine learning con Python. Costruire algoritmi per generare conoscenza*, ser. Guida completa. Apogeo, 2016. [Online]. Available: https://books.google.it/books?id=G7OvDAEACAAJ

[28] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, 2020.

[29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[30] J. Huang, V. Rathod, C. Sun, M. Zhu, A. K. Balan, A. Fathi, I. C. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3296–3297, 2016.

[31] models/research/object_detection at master · tensorflow/models · github. [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection

[32] Y. Li, H. Huang, Q. Xie, L. Yao, and Q. Chen, "Research on a surface defect detection algorithm based on mobilenet-ssd," *Applied Sciences*, vol. 8, no. 9, p. 1678, Sep 2018. [Online]. Available: http://dx.doi.org/10.3390/app8091678

[33] Free - flir thermal dataset for algorithm training | flir systems. [Online]. Available: https://www.flir.com/oem/adas/adas-dataset-form/

[34] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[35] Y. Uzun and M. Bilban, "Autonomous vehicle and augmented reality usage," *Journal of Civil Engineering and Management*, vol. 09, 12 2019.