République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Abderrahmane Mira

Faculté de Technologie

Département d'Automatique, Télécommunication et d'Electronique

Projet de Fin d'étude

Pour l'obtention du diplôme de Master

Filière : Télécommunications

Spécialité : Réseaux et Télécommunications

Thème

Intelligent Embedded Vision for Summarization of Multi-View Videos in IoT

Préparé par :

CHETTOUH Jugurtha

ABBACI Aissa

Dirigé par :

Mme. MEZZAH Samia

Examiné par :

Pr: MOKRANI Karim

Dr: AZNI Mohamed

Année universitaire : **2020/2021**

Acknowledgements

First of all, praise to God who has allowed us to find unsuspected strengths within ourselves and who has helped us to overcome difficulties and give color back to life.

Secondly, we would like to warmly thank Mme **MEZZAH Samia**, without whom this work would never have succeeded, a teacher whom we admire so much, and a supervisor who helped us, inspired and encouraged us, and with whom we established a relationship of trust. We are proud today to have been his disciples.

Finally, we would like to thank the members of the jury for the time they took to examine our work and for their suggestions and pertinent remarks.

Dedication

As a token of my deep gratitude, I dedicate this modest work

To my dearest parents

I would like to express my respect, my eternal love and my consideration for the sacrifices you have made for my education and my well-being. I will never thank you enough for having supported and accompanied me.

To my dear brothers and sisters

I express to you through this work, my feelings of brotherhood, tenderness and love.

To all my precious Friends who will recognize themselves.

And to all the people who are important to us.

Contents Table

Table of Figures

Table Of Abbreviations

VSS	Video Surveillance System
IoT	Internet of Things
SVS	Static Video Summary
DVS	Dynamic Video Summary
SVVS	Single View video Summarization
IoT	Industrial Internet of Things
LSTM	Long and Short-Term Memory
MVVS	Multi-Views video Summarization
MVV	Multi-Views Video
AI	Artificial Intelligence
NLP	natural-language processing
M2M	machine-to-machine
RAM	Random Access Memory
SBC	Single-Board Computer
OS	Operating System
USB	Universal Serial Bus
GPIO	General - Purpose Input/Output
HDMI	High-Definition Multimedia Interface
SSH	Secure Shell-to access
FTP	File Transfer Protocol
SVM	Support Vector Machines
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GAN	Generative Adversarial Network
DBN	Deep Belief Network
FC	Fully Connected
OCR	Optical Character Recognition
SSD	Single-Shot MultiBox Detector
SPP	Spatial Pyramid Pooling
R-CNN	Region-based Convolutional Neural Network
YOLO	You Only Look Once
HOG	Histogram of Oriented Gradients
SIFT	Scale Invariant Features Transform
LBP	Local Binary Pattern
LTP	Local Ternary Patterns

IOU	Intersection Over Union
ML	Machine learning
RPi	Raspberry Pi
CPU	Central Processing Unit
GPU	Graphics Processing Unit
AVR	Advanced Virtual RISC
PIC	Peripheral Interface Controller
UART	Universal Asynchronous Receiver Transmitter
I2C	Inter-Integrated Circuit, eye-squared-C
SoC	System On a Chip
BLE	Bluetooth Low Energy
CSI	Camera Serial Interface
DSI	Display Serial Interface
LAN	Local Area Network
PoE	Power Over Ethernet
MIPI	Mobile Industry Processor Interface
DPI	Parallel RGB Display
SPI	Serial Peripheral Interface
PCM	Pulse Code Modulation
PWM	Pulse Width Modulation
GPCLK	General Purpose Clock
RGB	Red Green Blue
SDIO	Secure Digital Input Output,
API	Application Programming Interface
ARM	Advanced RISC Machine
LXDE	Lightweight X11 Desktop Environment
RTOS	Real-Time Operating System
UEFI	Unified Extensible Firmware Interface
BIOS	Basic Input/Output System
SSD	Solid-State Drive
SDRAM	Synchronous Dynamic Random Access Memory
ROM	Read Only Memory
Sh	Bourne Shell
GNU	GNU is Not Unix
GUI	Graphical User Interface
DE	Desktop Environment
LKM	Loadable Kernel Modules
RTSP	Real-time Streaming Protocol
ZMQ	ZeroMQ
MPEG	Motion Picture Experts Group
FFMPEG	Fast Forward MPEG
MQTT	MQ Telemetry Transport
AMQP	Advanced Message Queuing Protocol
ROS	Route Orientation Scheme
Fps	Frames Per Second
CI	Computationally Intelligent
PNG	Portable Network Graphics
OOP	Oriented Object Programming

IDE	Integrated Development Environment
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
CoDec	Coder-Decoder
HTTP	Hypertext Transfer Protocol
WSGI	Web Server Gateway Interface
mDNS	Multicast Domain Name Server
DDNS	Dynamic Domain Name Server
VPN	Virtual Private Network
TLS	Transport Layer Security
SSL	Secure Socket Layer
DL	Deep Learning
JPEG	Joint Photographic Expert Group
TCP	Transmission Control Protocol

Introduction

Over the last few decades and with the increased demand for security, video based surveillance systems had gained a considerable importance. These systems have moved from traditional analogue video to digital Internet protocol (IP) video for better efficiency and reliability. However, the amount of video data makes it a big data problem that demands continuous development of more video analysis methods to extract the maximum of information automatically. Moreover, the scopes like prevention and intervention have led to the development of intelligent video surveillance systems capable of video processing competencies based on techniques of computer vision and artificial intelligence. Among one the most important video analytics, we find the video summary. Video Summarizing is very useful because it allows the extraction of meaningful information from large amounts of video data in the form of keyframes or video skims.

On the other hand, the emergence of the Internet of Things (IoT) and the computing on the edge nodes, has led to the appearance of many solutions that propose intelligent distributed video surveillance systems [1]. Hence, the intelligence of the system is distributed in multiple nodes, where each one can include a camera and a processing module that performs vision tasks, such as object detection and tracking activity, before sending the information to the central framework. For vision tasks. current detection algorithms include Deep Neural Networks (DNNs). To perform objet detection and classification based on DNNs, a high-end hardware system with high computing capability and low power consumption is required.

in this context, we propose an Intelligent video-surveillance system bases on embedded vision using low cost raspberry platform. The developed software use a YOLO light weighted network that has been selected for detecting objects of interest. This system can be easily installed and configured to work as a smart camera edge node in a video-surveillance system.

The dissertation is organized as follows. Chapter I present an overview of intelligent video surveillance systems in IoT environment. In chapter II, we recall principles of machine learning, Deep Learning and Computer Vision. Chapter III gives the architecture of the proposed intelligent embedded vision system with the description of the main hardware and software aspect. In chapter IV, we describe the software implementation of the proposed system for the generation of the video summary.

Chapter I

Overview of video analysis for surveillance systems in IoT

Chapter - I - Overview of Video Analysis for Surveillance Systems in IoT

I.1 Introduction

Video surveillance involves the act of observing scenes and looking for specific behaviors that are improper or that may indicate the emergence or the existence of security issues. A digital video surveillance system is a system capable of capturing images and videos that can be processed, compressed, stored or sent over communication networks. It can be used for nearly any environment. In this chapter we present un overview of video surveillance systems characteristics and methods used for the analysis of generated video and its automation for efficient information extraction.

I.2 Video surveillance system

Video Surveillance Systems (VSS) provide surveillance capabilities used in the protection of people, assets, and structures. These systems are often used to support comprehensive security systems by incorporating video coverage and security alarms for barriers, intrusion detection, and access control.

VSS uses many connected components with a variety of functions, features, and specifications to generate, transmit, display, and store video data. this system can be as simple as a camera connected to a video monitor or a larger complex system operated by professional security personnel and comprised of a number of components falling into several basic categories like:

- Cameras;
- Monitors;
- Switchers and multiplexers;
- Video recorders;
- power and lighting.

Figure I.1 provides typical surveillance system components. Many features exist within each of these categories that can satisfy an agency's operational requirements in the most challenging environments. The most complex surveillance systems may incorporate hundreds of cameras and sensors integrated into one overall security network [2].

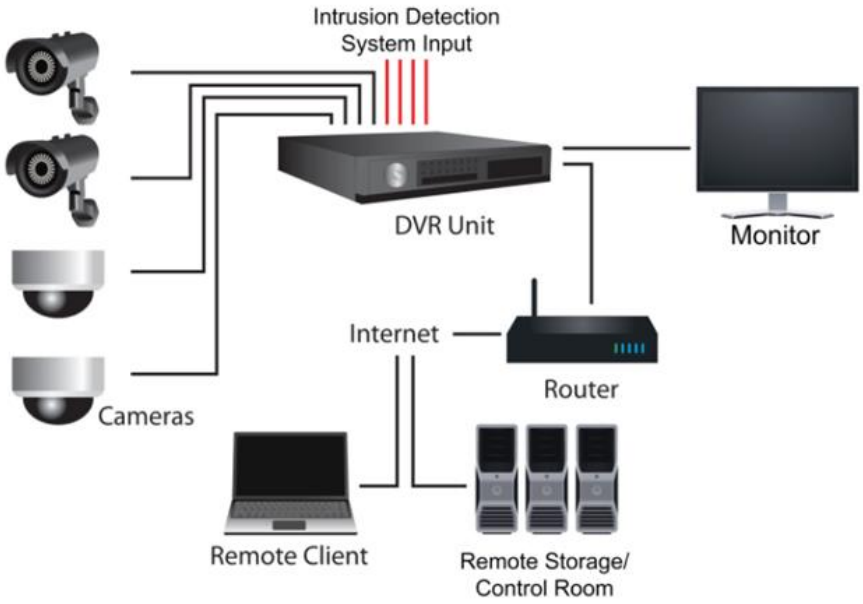

***Figure I.1**: Typical surveillance system components*

Video surveillance technologies continuously undergo feature refinements to improve performance in areas such as digital equipment options, data storage, component miniaturization, wireless communications, and automated image analysis. [3] As video surveillance technology has evolved, video transmission has progressed from analog to digital transmission. New cameras with Internet protocol (IP) capability transmit compressed video as digital data. A drawback of IP transmissions is that video places a high demand on a network's bandwidth, and the tradeoff may be image quality.

I.3 Intelligent automated video surveillance

Traditional video surveillance systems depend much on human interventions, and they are restrained to a simple On/Off switch, which results in thousands of hours of unusable video. This makes effective surveillance nearly impossible. Therefore, automated video surveillance replaced traditional systems by incorporating video analytics that uses computer algorithms to

monitor real-time video. Video analytics can help organizations become more efficient by automating part of the monitoring process and averting the time-consuming and tedious process of reviewing extensive quantities of stored video. However, with the increase of security concerns, the need for more effective video analytics approaches had emerge, that's why traditional automated surveillance systems had been complemented and even replaced by the advanced intelligent surveillance systems that enable very high monitoring accuracy by a few observers. In broad terms, advanced video-based surveillance could be described as an intelligent video processing technique designed to assist security personnel's by providing reliable real-time alerts and to support efficient video analysis for forensic investigations.

I.4 IoT application in video surveillance

Internet of Things (IoT) describes the network of physical object that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. it is network of smart devices that has its own computing capability, which are connected to form systems that collect, monitor, exchange and analyze data at industrial level.

IoT has replaced the traditional sensing of surrounding environments. Hence, intelligent video surveillance is an IoT based application as it uses Internet for various purposes. The IoT help to securely and remotely monitor facilities and public spaces in real-time with smart security and surveillance solutions. Video surveillance systems are a system of one or more intelligent video cameras capable of image analysis like object-detection and tracking connected on a network that send the captured video and audio information to a certain place. They are live monitored or transmitted to a central location for recording and storage. Intelligent video can provide the following improvements on previous security monitoring services [4]:

- Tracking a moving target;
- Automatic audio and visual detection of suspicious activity, which can trigger alarms and alert police station and business owners to potential threats;
- High-definition picture quality, as well as night vision technology triggered by motion sensors, meaning the system isn't running while nothing is happening at your location;
- Alerts notifying operators or field personnel;
- Ability to count people entering and leaving;
- Camera tampering detection;
- Vehicle license plate recognition.

I.5 Video analytics

Video analytics encompasses mainly the below-mentioned tasks:

1. Storage optimization: Storage optimization is realized based on the motion detection or summary generation. The video management systems decide to store the video when any motion/ interesting object or event is spotted in the observed scene or else the video is either not stored or is stored at a lower frame rate or a lower resolution to save storage space. Cameras may capture long durations of inactivity when placed in buildings when they are locked, staircases, etc. This application helps in reducing the consumption of storage as compared to continuous recording.
2. Identify threatening events: Video analytics can also be used to identify threatening events to pro-actively identify any lapse in security incidents, be alert and to stop them; for example, license plate recognition, perimeter violation, abandoned objects detection, and people counting.

I.6 Video summarization

Video summarization technologies aim to create a concise and complete summary video by compressing and selecting the most informative parts of the video content, it is an automatic technique for extracting significant information from big video data in the form of keyframes (still images) or video skims (moving images). It investigates the input video for different events, informative frames, then generates a summary that is representative of the whole video. Video summarization helps users to navigate through a large sequence of videos and retrieve ones that are most relevant to the query. Video summarization can be categorized into four forms:

- Static Video Summary;
- Dynamic Video Summary;
- Single View Summary;
- Multi-View Video Summary.

I.6.1 Static Video Summary (SVS)

Static Video Summary are usually presented as a storyboard; it consists of keyframes that represent mainly video content. It takes into account the visual information but ignores the audio message [4]. The advantage of SVS is that the keyframes sets are not restricted by any timing or synchronization issues, and therefore, they offer much more flexibility in terms of organization for browsing and navigation purposes, in comparison to a strict sequential display of video skims.

I.6.2 Dynamic Video Summary (DVS)

Dynamic Video Summary is a video clip that combines image, audio and text information, compared to the dynamic video summary, the static video summary is easier to navigate and reduces the computational complexity for the recovery and analysis of the video, but the dynamic summary has the option of including audio and movement elements that potentially enhance both expressiveness and information from abstraction.

Figure I.3 present steps used to generate a video skim from the original video, first, we perform scene & audio segmentation, then we merge them after select the most informative clips.

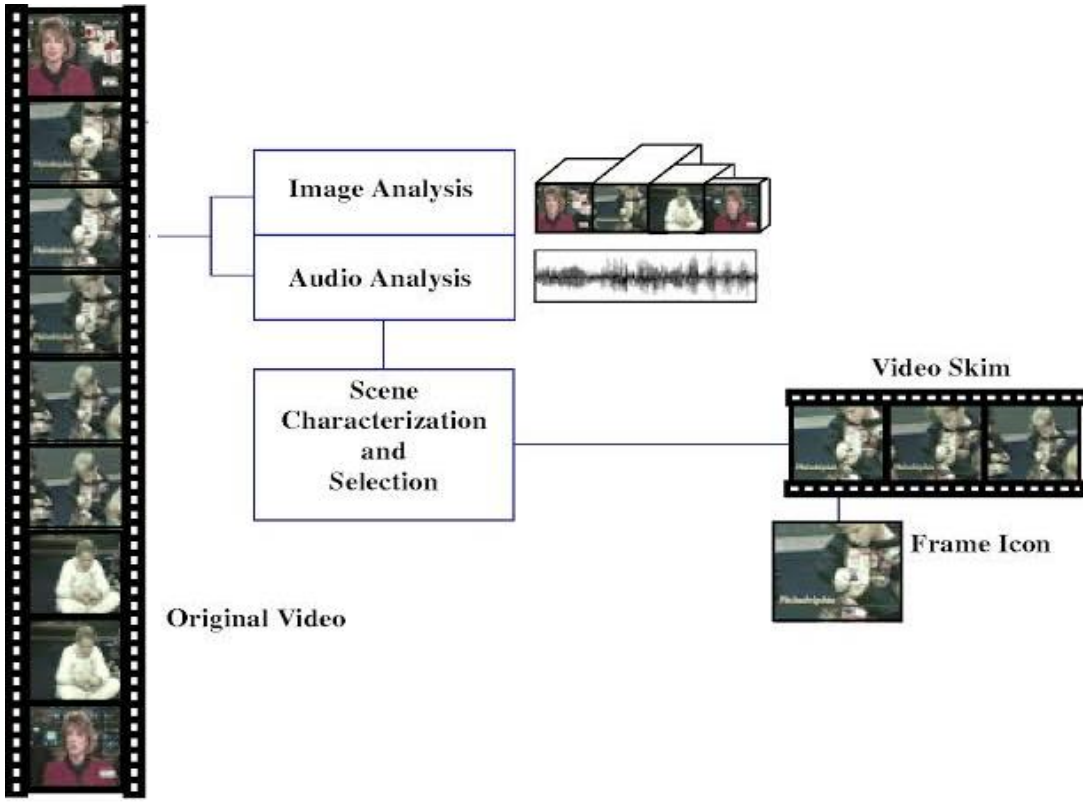

***Figure I.2**: Skim video for drastic reduction in viewing time without loss in content*

One advantage of a video skim method over keyframes method, is that it enhances the amount of information conveyed by the summary. In addition, it's often more entertaining and interesting to watch a skim than a slide shows of keyframes.

I.6.3 Single View Video Summarization (SVVS)

SVVS generates summary from a single input video, covering only one view at a time. A single camera for smart industries in IoT network has a limited coverage that cannot fully exploit the overall environment synchronously.

I.6.4 Multi-Views Video Summarization (MVVS)

MVVS generates summary from a multiple input videos, it refers to the problem of summarizing multi-view videos into informative video summaries, usually presented as dynamic video shots coming from multi-views, by considering content correlations within each view and among multiple views.

As opposed to SVVS, MVVS is challenging because it has both inter and intra-view correlations to be considered while generating summary. Another big challenge in MVVS is variation of light conditions among different views. Furthermore, synchronization among different views makes the problem of MVVS a difficult one. The basic flow of MVVS comprises of three steps: Pre-processing, Features extraction and summary generation [4]:

1. ***Pre-processing Multi-View Video (MVV)***

The first step in MVVS flow suppresses the Multi-View Video through several redundancy removal techniques such as shots segmentation (uniform or variable length) and video splitting based on the shots boundary.

2. ***Features extraction***

Which mainly includes objects detection, and tracking, they are considered to be the second prerequisite step followed by different MVVS methods.

3. ***Summary generation***

Final step of MVVS involves summary generation from the extracted features through various machine learning or template matching algorithms.

I.7 Conclusion

In this chapter we introduced Video surveillance, then we had a look about some IoT applications in Video surveillance, after that we explained video summarization along with analytics. computer vision,

in the next chapter, we will recall Artificial intelligence, machine learning, deep learning then we will end up with computer vision the most widely used real time object detector YOLO.

Chapter II

Deep learning for computer vision

Chapter - II - Deep learning for computer vision

II.1 Introduction

Artificial Intelligence, Machine Learning and, Deep Learning are the latest keywords of this century. Their wide range of applications has changed the facets of technology in every field, ranging from Healthcare, Manufacturing, Business, Education, Banking, Information Technology etc. In this chapter we present Deep learning for computer vision, we first introduce artificial intelligence, then machine learning and its different types (supervised, unsupervised and reinforcement learning), after that, we will recall deep learning which is based on Artificial neural network (ANN), finally we will end up by presenting CNN for embedded vision and the fastest object detection algorithm (YOLO).

II.2 Artificial Intelligence

Artificial intelligence, commonly referred to as AI, is the process of imparting data, information, and human intelligence to machines. The main goal of AI is to develop self-reliant machines that can think and act like humans. These machines can mimic human behavior and perform tasks by learning. Most of the AI systems simulate natural intelligence to solve complex problems. Amazon Echo is a good example of Artificial Intelligence (Figure II.1).

Figure II.1: *Artificial intelligence example - Amazon Echo*

II.3 Machine Learning

Machine learning is a field of study which allows machines (computers) to learn from data or experience and make a prediction based on the experience. It enables the computers or the machines to make data-driven decisions rather than being explicitly programmed for carrying out a certain task. These programs or algorithms are designed in a way that they learn and improve over time when are exposed to new data. Machine learning accesses vast amounts of data (both structured and unstructured) and learns from it to predict outcomes accurately. It employs various approaches to teach computers in order accomplish tasks where no fully satisfactory algorithm is available [5].

II.3.1 Types of Machine Learning

Machine learning algorithms are classified into three main categories: Supervised Learning, Unsupervised Learning and Reinforcement Learning.

II.3.1.a Supervised Learning

Supervised learning category is concentrated on mapping patterns by establishing the relationship between variables and known outcomes while working with labelled datasets. Simply saying, it's like having an input variable (x) and output variable (y) and use an algorithm to make it learn to establish a mapping function between the input and output.

With supervised learning, the machine already knows the output of the algorithm before it starts working on it or learning it. This means that if the process goes haywire and the algorithms come up with results completely different than what should be expected, then the training dataset will guide the algorithm back towards the right path. Using this method of learning,

systems can predict future outcomes based on past data. It requires that at least an input variable be given to the model for it to be trained.

Figure II.2 present a basic supervised learning process. We input Raw Data to the algorithm, this algorithm will learn then make predictions using a given dataset and is corrected by the "supervisor". After generating a trained model, the model can then accept new input data; process it then give desired outputs according to what it's trained for.

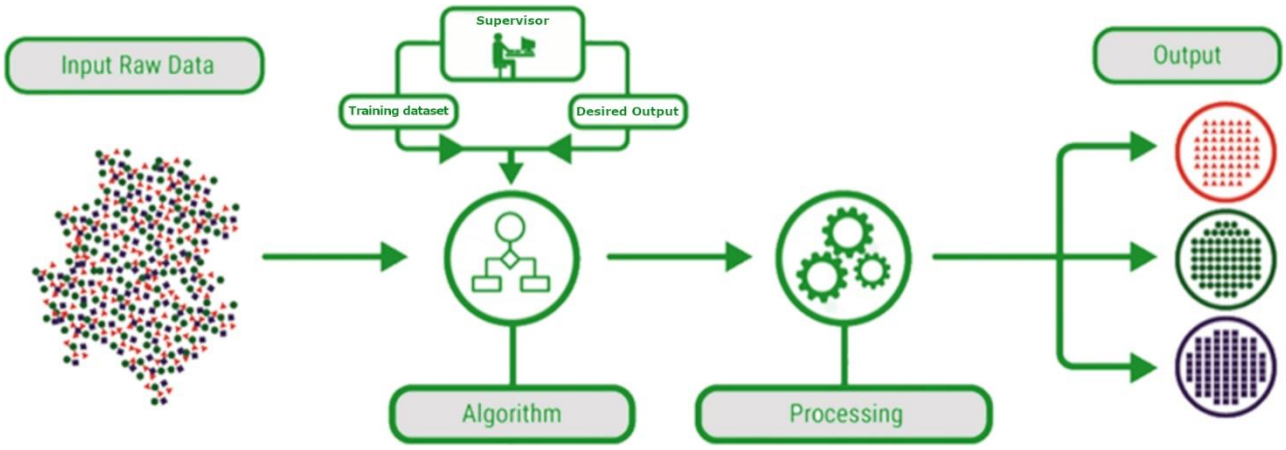

Figure II.2: *Supervised learning process*

Supervised learning can further be divided into two types of tasks:

- ***Regression*****:** Regression problem is when the output variable is continuous and real value. For example, price, weight, etc.
- ***Classification*****:** Classification is a problem when the output variable is a category (ex. mask, no-mask, person, car…etc.) and can be used in applications such as speech recognition, handwriting recognition, biometric identification, document classification…etc.

Some examples of supervised learning algorithms include: linear regression, logistic regression, support vector machines (SVM), Naive Bayes, and decision tree. We can see in Figure II.3 the list of most widely used Regression and classification algorithms.

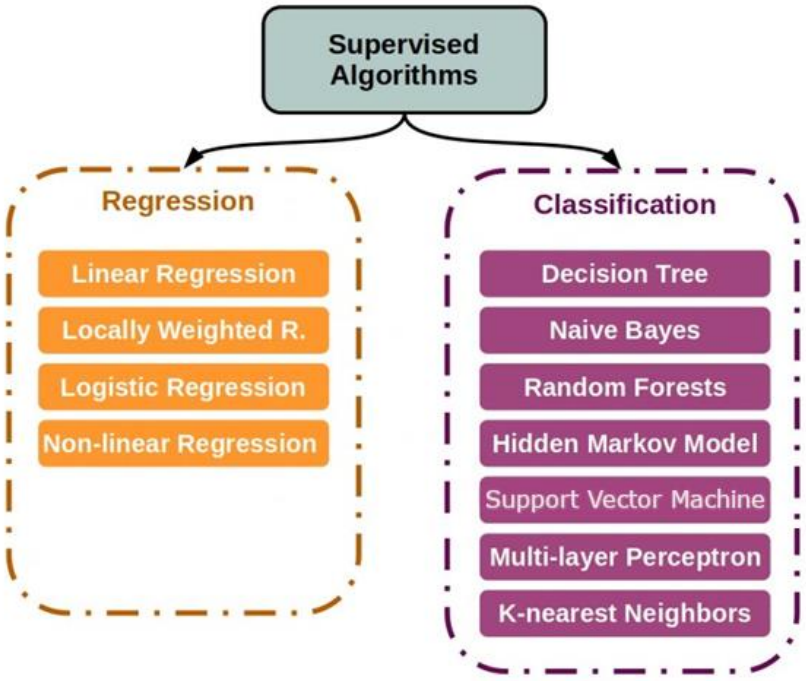

***Figure II.3**: Regression & Classification algorithms*

II.3.1.b Unsupervised Learning

Unsupervised Learning is a machine learning technique in which the users do not need to supervise the model. Instead, it allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with the unlabelled data (Figure II.4). The systems are able to identify hidden features from the input data provided (interpretation). Once the data is more readable, the patterns and similarities become more evident. After generating a trained model, the model can now cluster any new data accordingly.

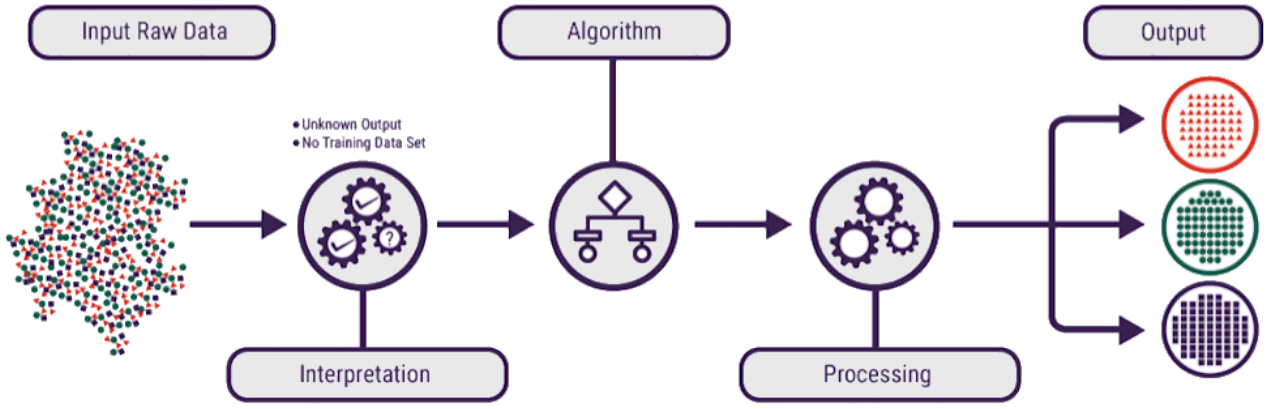

***Figure II.4:** Unsupervised learning process*

Unsupervised learning can further be divided into two types of tasks:

- ***Clustering:*** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- ***Association*****:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Some examples of unsupervised learning algorithms include: k-means clustering, hierarchical clustering, and anomaly detection.

II.3.1.c Reinforcement Learning

Reinforcement learning, in the context of artificial intelligence, is a type of dynamic programming that trains algorithms using a system of *reward* and *punishment*. A reinforcement learning algorithm, or agent, learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without human intervention by maximizing its reward and minimizing its penalty.
Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience. Figure II.5 shows how reinforcement learning works to train itself with rewards and penalties according to given policy.

The main points we should consider in Reinforcement learning:

- ***Input:*** The input should be an initial state from which the model will start;
- ***Output:*** There are many possible output as there are variety of solution to a particular problem;
- ***Training:*** The training is based upon the input, the model will return a state and the user will decide to reward or punish the model based on its output;
- The model keeps continue to learn;
- The best solution is decided based on the maximum reward.

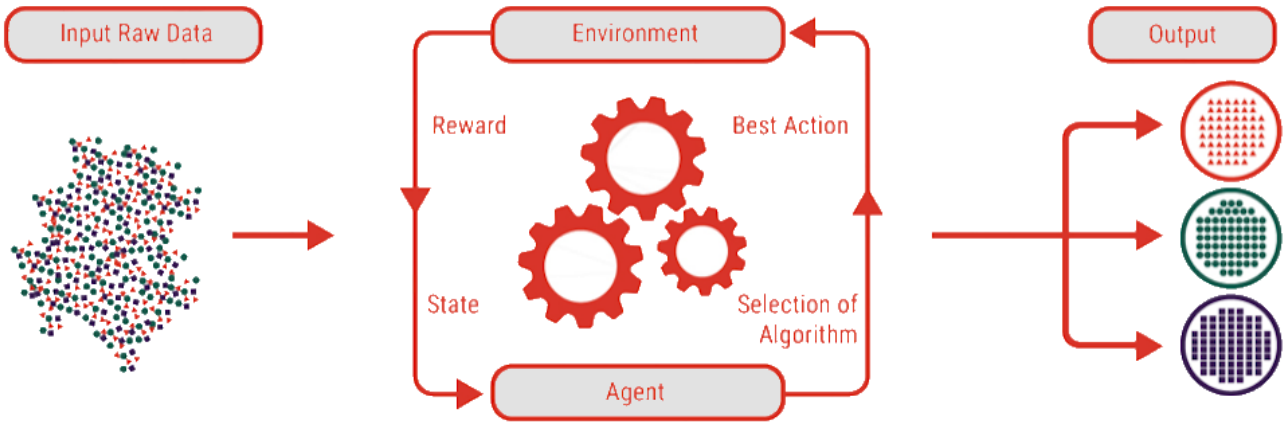

***Figure II.5**: Reinforcement learning process*

Examples of reinforcement learning algorithms include: Q-learning and Deep Q-learning Neural Networks.

II.3.2 Machine learning applications

Figure II.6 shows different applications according to the type of machine learning:

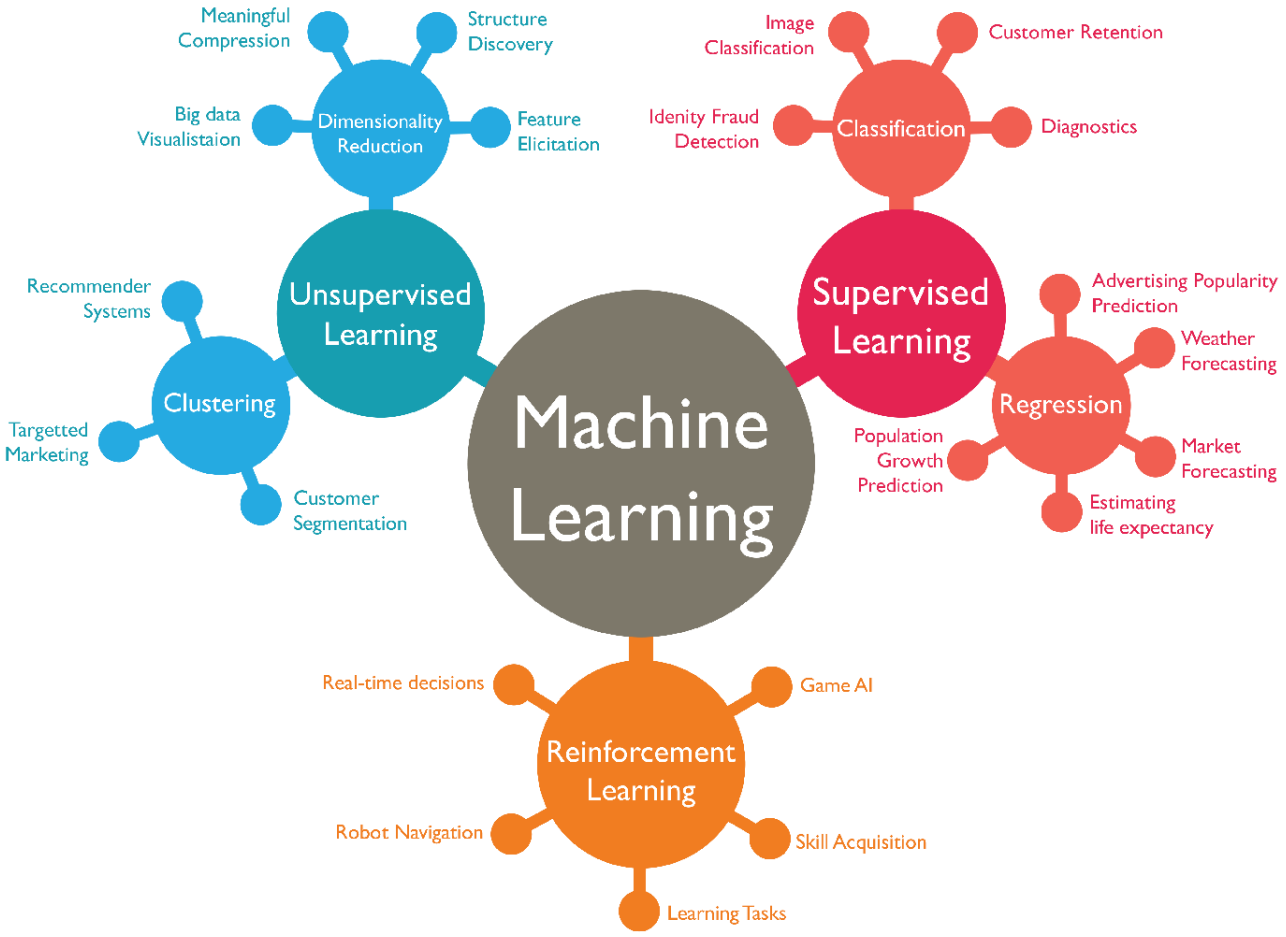

***Figure II.6**: Different Machine-Learning applications according to their types*

II.4 Deep Learning

Deep learning is a subset of machine learning which itself is a subset of Artificial intelligence (Figure II.7), it deals with algorithms inspired by the structure and function of the human brain. Deep learning algorithms work with an enormous amount of both structured or unstructured data and its core concept lies in artificial neural networks, which enable machines to make decisions.

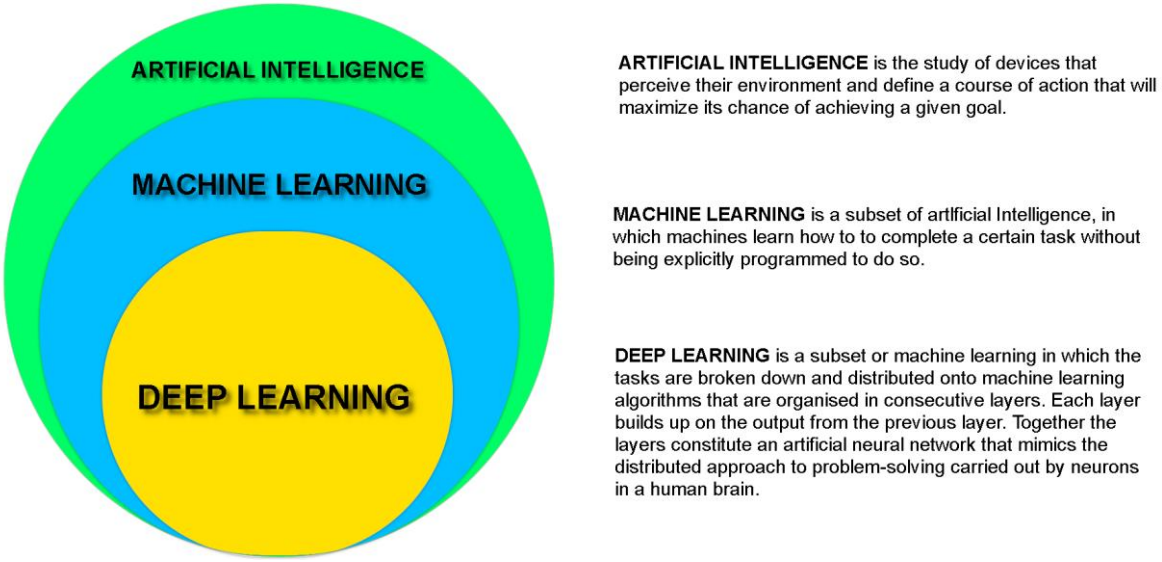

***Figure II.7**: Artificial intelligence VS Deep learning VS machine learning*

Artificial neural network (ANN) is the piece of a computing system designed to simulate the way the human brain analyzes and processes information. It is the foundation of AI and solves problems that would prove impossible or difficult by human or statistical standards. Figure II.8 describe a simple artificial neural network structure:

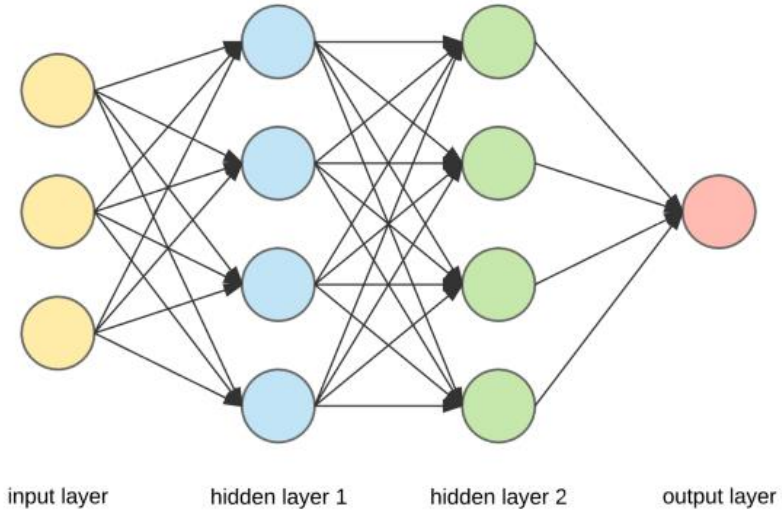

***Figure II.8** : Basic Artificial Neural Network structure*

The ANN is constructed from 3 types of layers:

Input layer: initial data for the neural network.

Hidden layers: intermediate layer between input and output layer and place where all the computation is done.

Output layer: produce the result for given inputs.

II.4.1 Deep Networks types

There are many types of neural networks in deep learning which are used for different purposes, but 4 of them are considered as major types of deep neural network:

- **Convolutional Neural Network (CNN) -** CNN is a class of deep neural networks most commonly used for image analysis.
- **Recurrent Neural Network (RNN)** - RNN uses sequential information to build a model. It often works better for models that have to memorize past data.
- **Generative Adversarial Network (GAN)** - GAN are algorithmic architectures that use two neural networks to create new, synthetic instances of data that pass for real data. A GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers.
- **Deep Belief Network (DBN)** - DBN is a generative graphical model that is composed of multiple layers of latent variables called hidden units. Each layer is interconnected, but the units are not.

II.4.2 Deep Learning Applications

There are many Deep Learning applications, some of them are:

- Caption bot for captioning an image;
- Object detection and face recognition;
- Self-driving cars;
- Voice search and virtual assistants;
- Machine translation;
- Colorization of Black and White Images;
- Game playing **AI** (Open Ai dota bot, google brain alpha go);
- Real-time object recognition in the image (Google lens).

II.5 Convolutional Neural Network (CNN)

A Convolutional Neural Network is a type of artificial neural network used mainly in image recognition and processing that is specifically designed to process pixel data. CNN is one of the most popular algorithms for deep learning with images and videos.

II.5.1 Basic CNN architecture

Like other neural networks, a CNN is composed of an input layer, an output layer, and many hidden layers in between (Figure II.9):

A- Feature Detection Layers: in the basic architecture, these layers perform one of 2 types of operations on the data: *convolution*, *pooling*. Convolution puts the input images through a set of convolutional filters, each of which activates certain features from the images. Pooling simplifies the output by performing nonlinear down-sampling, reducing the number of parameters that the network needs to learn about. These operations are repeated over tens or hundreds of layers, with each layer learning to detect different features.

B- Classification Layers: After feature detection, the architecture of a CNN shifts to classification. The next-to-last layer is a fully connected layer (FC) that outputs a vector of K dimensions where K is the number of classes that the network will be able to predict. This vector contains the probabilities for each class of any image being classified. The final layer of the CNN architecture uses the calculated class probabilities to provide the classification output.

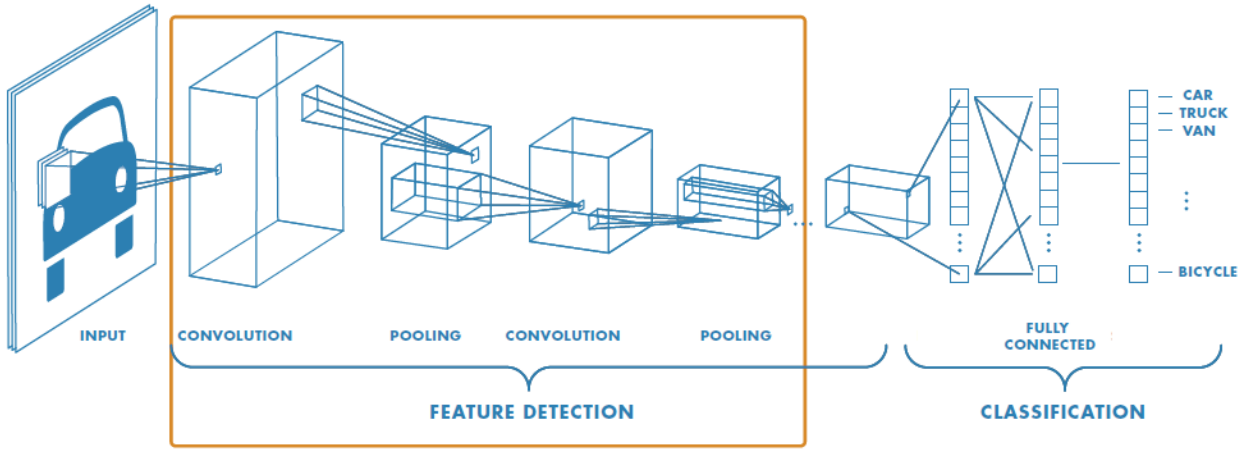

***Figure II.9** : Basic CNN structure [6]*

II.5.2 Learning Process

A neural network is made up of neurons connected to each other at the same time, each connection of the neural network is associated with a weight that dictates the importance of this relationship in the neuron when multiplied by the input value as shown in Figure II.10. Each

neuron has an activation function that defines the output of the neuron. The activation function is used to introduce non-linearity in the modeling capabilities of the network.

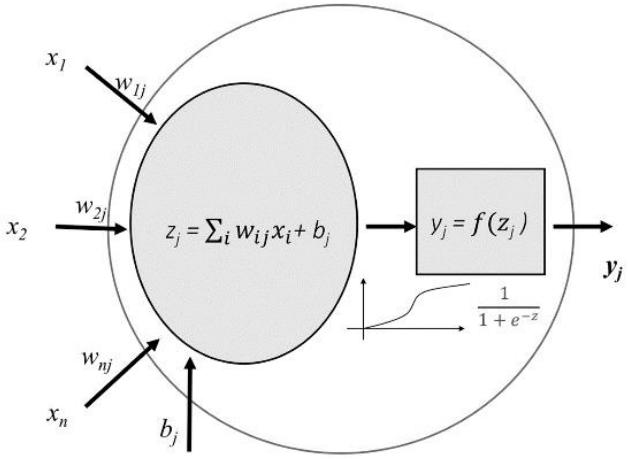

Figure II.10: *A presentation of neuron : "wij" weights, "bj" biases, "xn" inputs, and "yi" output*

The most essential element of Deep Learning is learning the values of parameters "wij". The learning process in a neural network is an iterative process consisting of two phases (Figure II.11):

- The first phase of forward propagation occurs when the network is exposed to the training data and these cross the entire neural network for their predictions (labels) to be calculated. That is, passing the input data "*xn"* through the network in such a way that all the neurons apply their transformation to the information they receive from the neurons of the previous layer and sending it to the neurons of the next layer. When the data has crossed all the layers, and all its neurons have made their calculations, the final layer will be reached with a result of label prediction for those input examples.
- The second phase back propagation is based on the use of a loss function to estimate the loss (or error) and to compare and measure how good/bad neural prediction result was in relation to the correct result (label). The error is propagated backwards to all the neurons in the hidden layer that contribute directly to the output. Therefore, as the model is being trained, the weights of the interconnections of the neurons will gradually be adjusted until good predictions are obtained.

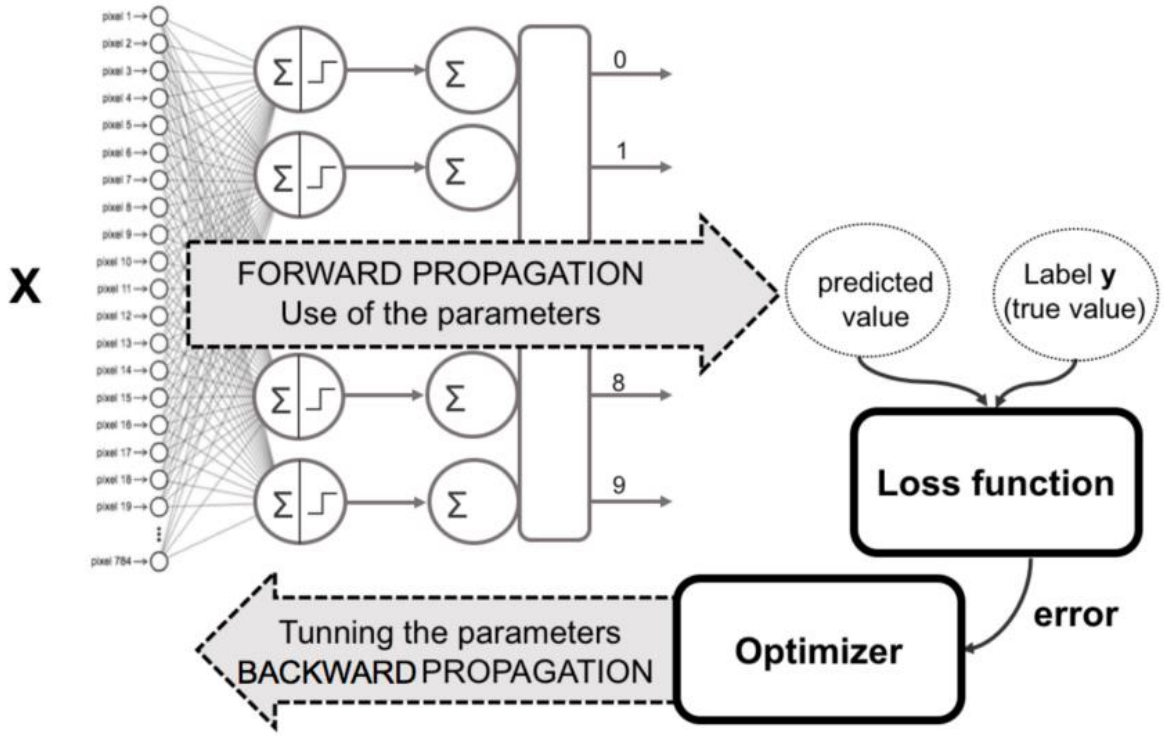

***Figure II.11**: Back-propagation and forward-propagation process*

After performing back-propagation, weights of neurons are adjusted by minimizing errors with optimizers algorithms (like gradient descent, stochastic gradient descent, momentum, AdaDelta, Adam…etc.), for case of gradient descent method, it adjusts the weights in small increments by calculating the derivative (or gradient) of the loss function (Figure II.12). This is done in general in batches of data in the successive iterations (epochs) of all the dataset that we pass to the network in each iteration.

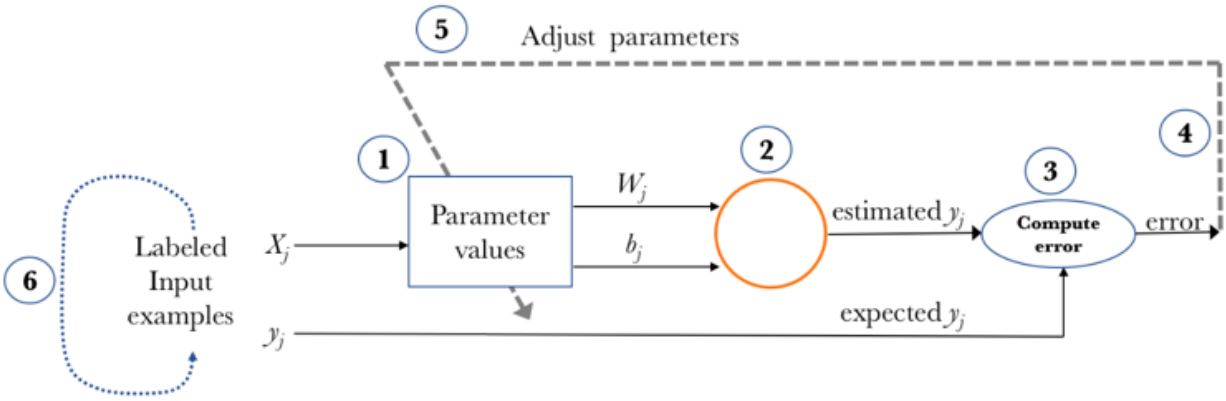

***Figure II.12**: Learning algorithm steps*

II.6 Computer Vision

Computer vision is a field of study focused on the problem of helping computers to see (interpret images). Typically, this involves developing methods that attempt to reproduce the capability of human vision by automating the extraction of information from images. Those Information can be anything from 3D models, camera position, object detection and recognition to grouping and searching image content [7]. It is a multidisciplinary field that could broadly be called a subfield of artificial intelligence and machine learning, which may involve the use of specialized methods and make use of general learning algorithms.

II.6.1 Computer Vision and Image Processing

Computer vision is distinct from image processing. Image processing is the process of creating a new image from an existing image, typically simplifying or enhancing the content in some way. It is a type of digital signal processing and is not concerned with understanding the content of an image. A given computer vision system may require image processing to be applied to raw input, e.g. pre-processing images.

Examples of image processing include normalizing photometric properties of the image (brightness or color), cropping the bounds of the image (centering an object in a photograph) and removing digital noise from an image (digital artifacts from low light levels).

II.6.2 Challenges of Computer Vision

Helping computers to see is such a challenging problem because of the complexity inherent in the visual world. A given object may be seen from any orientation, in any lighting conditions, with any type of occlusion from other objects, and so on. A true vision system must be able to "*see*" in any of an infinite number of scenes and still extract something meaningful. Computers work well for tightly constrained problems, not open unbounded problems like visual perception.

II.6.3 Computer Vision for Video Surveillance

The surveillance industry is one of the early adopters of image processing techniques and video analytics. Video analytics is a special use case of computer vision that focuses on processing hours of video footage with the ability to automatically detect and identify predefined patterns in real world situations with hundreds of use cases.

The first video analytics tools use handcrafted algorithms that identify specific features in images and videos (Figure II.13). They were accurate in laboratory settings and simulation environments. However, the performance quickly dropped when the input data, like lighting

conditions and camera views, deviated from design assumptions witch made them not robust enough. Handcrafted features such as Histogram of Oriented Gradients (HOG), Scale invariant Feature Transform (SIFT), Local Binary Pattern (LBP), Local Ternary Patterns (LTP). These handcrafted features produce high dimensional feature vectors obtained by the aggregation of small local patches of subsequent video frames for motion and appearance information. However, these high dimensional feature vectors do not scale well for large scale video processing in uncontrolled environment.

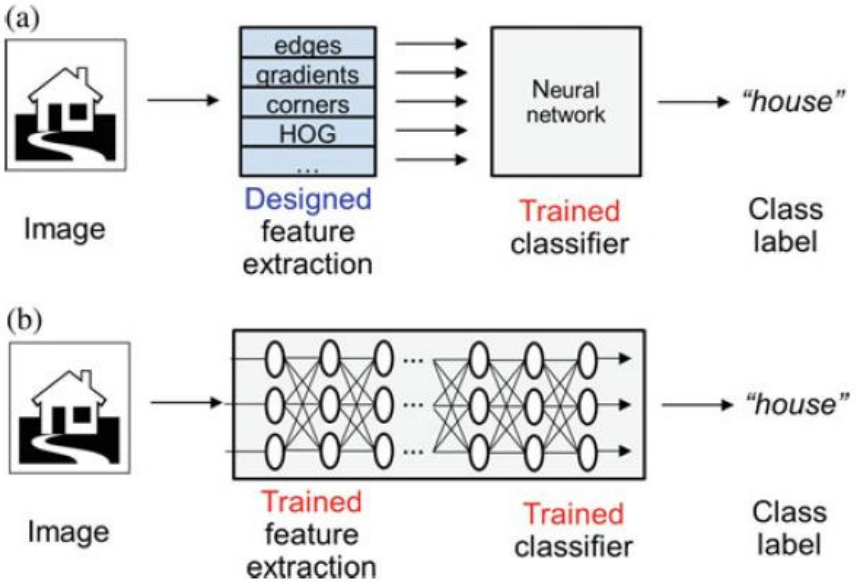

***Figure II.13**: Comparing the (a) classical approach to machine learning using handcrafted fetaures to the (b) deep learning operating on raw inputs.*

Recently, Video analytics solutions based on deep learning models like CNN serve as the foundation for cutting-edge analytics systems used in smart cities and real-time applications. Despite enormous effort in developing automated systems, current surveillance systems are not entirely capable of autonomously analyzing complex event from observed scene. To address this problem, independent work on several areas such as object tracking, behavior understanding, object classification, summarization and motion segmentation are combined to form a composite video analytic framework for video surveillance.

II.6.3.a Computer Vision tasks

Many popular computer vision applications involve trying to recognize things in images; for example:

- **Object Classification**: What broad category of object is in this image?
- **Object Identification**: Which type of a given object is in this image?
- **Object Verification**: Is the object in the image?
- **Object Detection**: Where are the objects in the image?
- **Object Landmark Detection**: What are the key points for the object in the image?
- **Object Segmentation**: What pixels belong to the object in the image?
- **Object Recognition**: What objects are in this image and where are they?

II.6.3.b Object Detection algorithms

Object detection is very important tool for video surveillance analytics that involves the detection of various objects like people, cars, animals, weapons...etc. Well-researched domains of object detection include face detection and pedestrian detection. Some of widely used object detection algorithms are:

- Region-based Convolutional Neural Networks (R-CNN);
- Single Shot Detector (SSD);
- Spatial Pyramid Pooling (SPP-net);
- YOLO (You Only Look Once).

II.7 Embedded vision with YOLO Network

YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). YOLO combines what was once a multi-step process, using a single neural network to perform both classification and prediction of bounding boxes for detected objects. As such, it is heavily optimized for detection performance and can run much faster than running two separate neural networks to detect and classify objects separately. It does this by repurposing traditional image classifiers to be used for the regression task of identifying bounding boxes for objects, the algorithm requires only a single forward propagation through a neural network to detect objects. This means that objects detection in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

Processing images with YOLO is simple and straightforward and can be divided into 3 main steps like shown in Figure II.13:

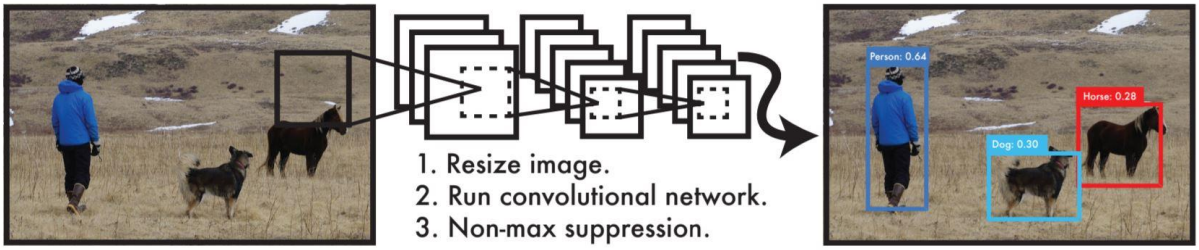

***Figure II.13**: YOLO Detection System [8]*

1. Resizes the input image to 448 × 448;
2. Runs a single convolutional network on the image;
3. Thresholds the resulting detections by the model's confidence.

YOLO is refreshingly simple, a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO algorithm works using the following three techniques:

- Residual blocks;
- Bounding box regression;
- Intersection Over Union (IOU).

II.7.1 Residual blocks

First, the image is divided into various grids (Figure II.14). Each grid has a dimension of S x S.

***Figure II.14**: How an input image is divided into grids in YOLO*

In Figure II.14, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

II.7.2 Bounding box regression

A bounding box is an outline that highlights an object in an image. Every bounding box in the image consists of the following attributes:

- Width (*bw*);
- Height (*bh*);
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter "*c*";
- Bounding box center (*bx*, *by*).

Figure II.15 shows an example of a bounding box. The bounding box has been represented by a yellow outline.

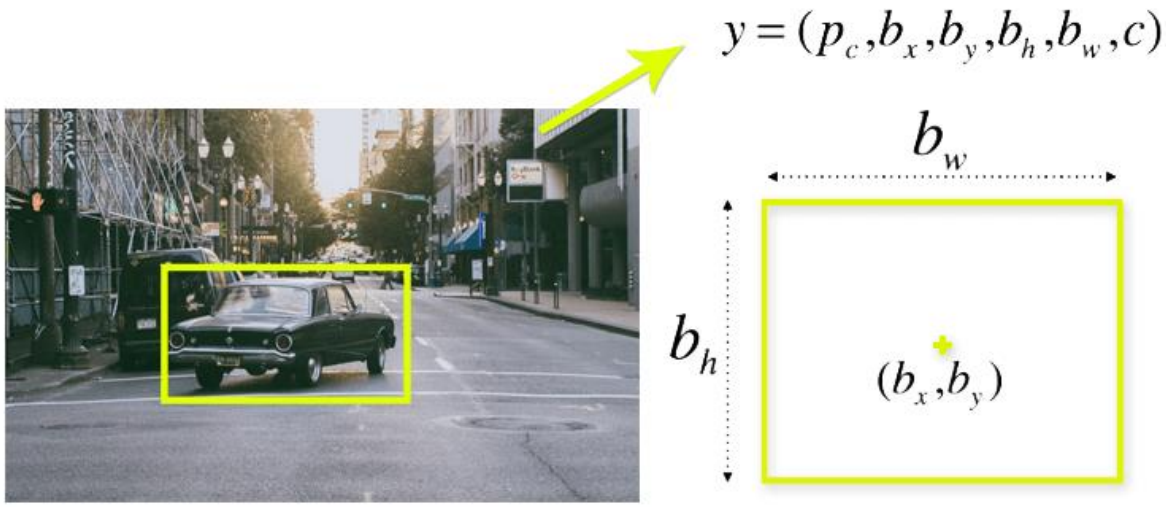

***Figure II.15**: An image shows an example of a bounding box in yollow color*

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. Figure II.15 presents the probability of an object appearing in the bounding box.

II.7.3 Intersection over union (IOU)

Intersection Over Union is so effective in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to **1** if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box [9].

Figure II.16 provides a simple example of how IOU works. There are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

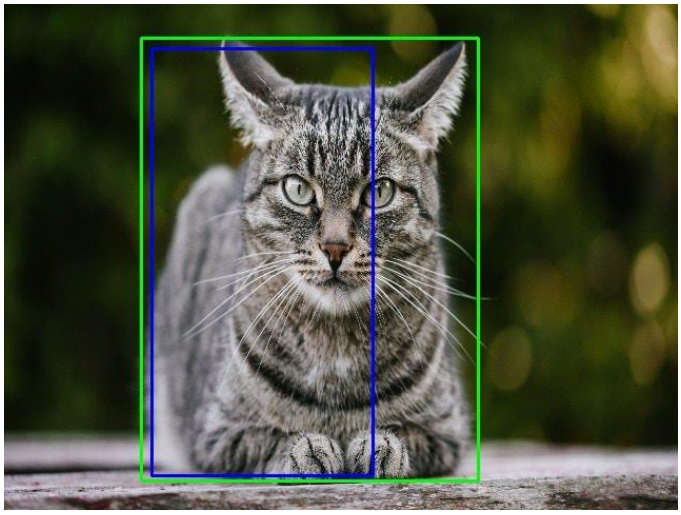

***Figure II.16**: An example IOU operation*

II.7.4 Combination of the three techniques

As we can see in Figure II.17 after combining the 3 techniques to produce the final detection results.

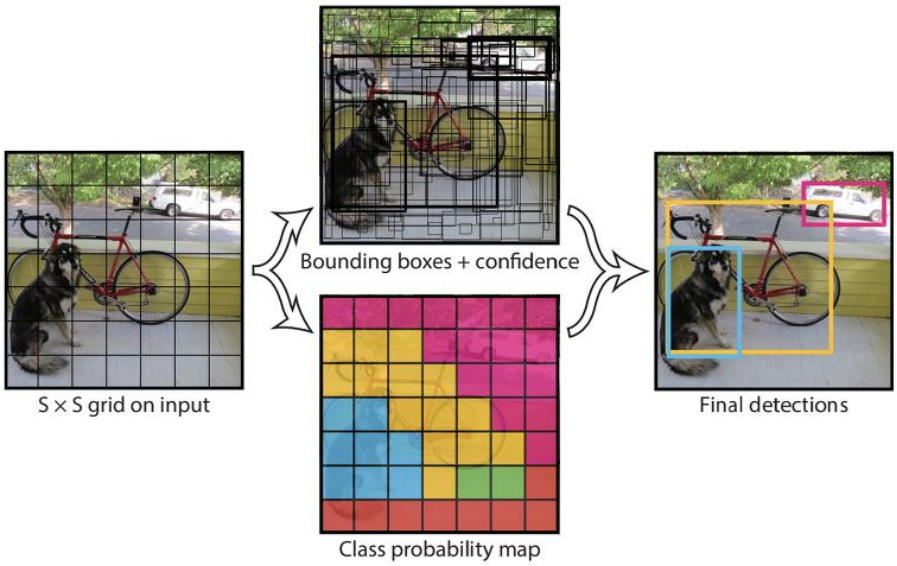

***Figure II.17**: Combination oh the 3 techniques to produce the final detection in YOLO*

First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object. For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single CNN [9].

IOU ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly.

For example, the car is surrounded by the pink bounding box while the bicycle is surrounded by the yellow bounding box. The dog has been highlighted using the blue bounding box.

II.8 Conclusion

In this chapter, we presented briefly Machine-Learning including and its different types: supervised, Unsupervised and reinforcement learning, after that we presented Deep learning with its types and some of its applications. then, we introduced computer vision; finally, we ended with explanation of YOLO algorithm for object detection.

In the next chapter, we will describe the proposed embedded vision system implementation which consist in the different hardware & software components that include preparation of the development environment by setting/tweaking the Linux configurations and installing required tools and libraries used in this project.

Chapter III

Intelligent Embedded Vision based MVS System Description

Chapter - III - Intelligent Embedded Vision based MVS System Description

III.1 Introduction

The objective of our project is studying and realizing intelligent components to integrate into application, then for analyzing surveillance videos to extract relevant information and generate summary. This component is based on the use of an embedded CNN. Our method uses an IoT network containing smart clients and master devices with built-in cameras to capture multi-view video data.

III.2 System Description

Our project approach is inspired from [4] presenting an Intelligent Embedded vision for MVS in IIoT, however the code is completely rewritten from scratch and enhanced to fits our need.
This project uses an IoT network containing smart devices, Raspberry Pi (clients and master) with embedded cameras to capture multi-view video (MVV) data. Each client Raspberry Pi (RPi) detects target in frames via light-weight CNN model, the system detects targeted object according to a selected YOLO model to generate alert in the IoT network (in case it's suspicious). The frames of each client RPi are encoded/encrypted and transmitted to master RPi for final MVS, the data communication are done through a VPN network for higher security and can be accessed via any device that support HTTP protocol. Our project can also be used in industrial environments for various applications such as security and smart transportation and can be proved beneficial for saving resources.

The overall scenario of the proposed framework in IoT setup is presented in Figure III.1

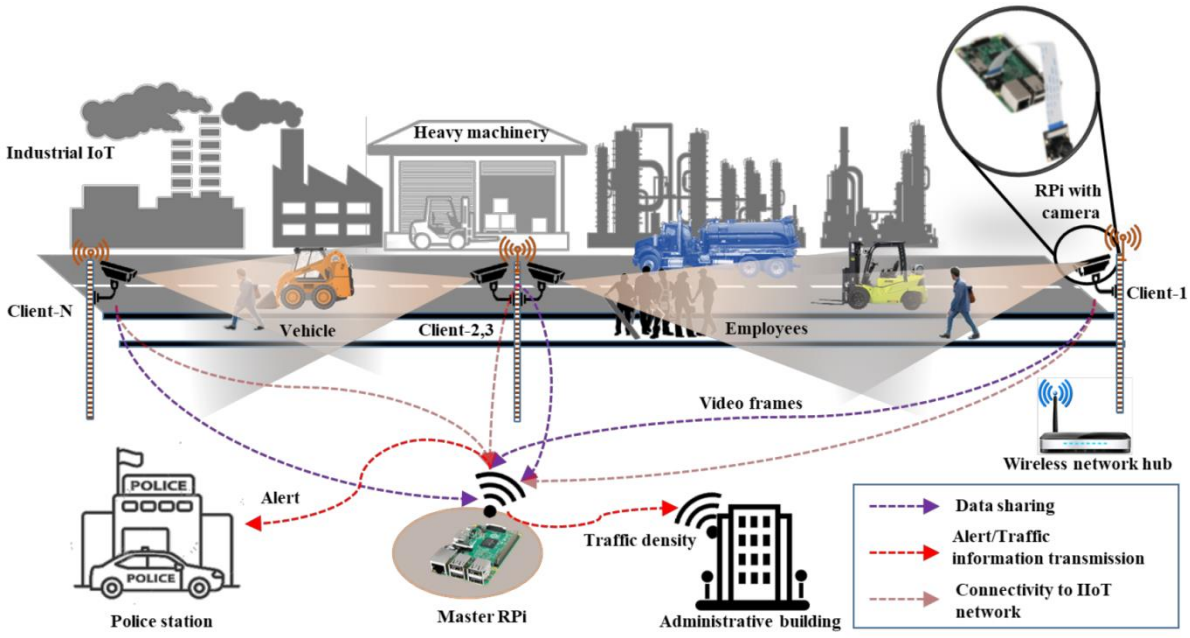

***Figure III.1**: Sample scenario for IoT connected devices (RPi's) in smart industries.*

Each Raspberry-Pi client detects targets (according to the implemented object detection model) then send the captured keyframes to the master Raspberry-Pi for the final Multi-View Summarization (MVS) process.

The proposed MVS system is based on single board computers (SBC) components, which are complete computers built on a single circuit board, with microprocessor, memory, input/output (I/O) and other features required of a functional computer, they are commonly made as development or educational systems. For our project, we chose the most used SBC in IoT applications which is the Raspberry Pi4. Raspberry Pi4 is an ARM-based SBC and a powerful tool when it comes to AI and ML. Moreover, Raspberry processing capabilities matched with a small form factor and low power requirements, make it an adapted device for smart robotics objects and embedded projects which requires significant processing power, energy efficient and low power consumption at cheap price.

III.3 Raspberry platform

The Industrial Raspberry Pi offers a versatile set of tools for solving almost any automation challenge and it operates in the open source ecosystem, it runs Linux (a variety of distributions), and its main supported operating system, Raspbian OS.

III.3.1 Raspberry Pi boards

The Raspberry Pi (RPi) is a series of single-board computers. They are low-cost, high-performance and the size of a credit card. It was developed in the UK by the Raspberry Pi Foundation. The Raspberry Pi Foundation's goal is to "advance the education, particularly in the field of computers, computer science and related subjects".

The Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects and industries, from music machines and parent detectors to weather stations and embedded vision for intelligent video surveillance.

III.3.1.a Raspberry-Pi use cases

The major components of a Raspberry Pi boards, is the Broadcom System-on-Chip (SoC) that include a central processing unit (CPU), a graphics processing unit (GPU), memories and various digital, analogue and mixed signal circuits such as timers, USB controller, PCM/I2S, SPI/I2C et UART.

The RPi is adapted to projects of high-level software integration and low level electronics. The major advantage the RPi and other embedded Linux devices have over more traditional embedded systems, such as the Arduino, PIC, and AVR microcontrollers, is apparent when you leverage the Linux OS for your projects. Linux provides us with device driver support for many USB peripherals and adapters, making it possible for us to connect cameras, Wi-Fi adapters, and other low-cost consumer peripherals directly to our platform without the need for complex/expensive software driver development.

The RPi is also an excellent device for playing high-definition video and this is due to its Broadcom BCM2835/6/7 processor that was designed for multimedia applications, moreover, it has a hardware implementation of H.264/ MPG-4 and MPG-2/VC-1 (via additional license) decoders and encoders witch make it ideal for Computer Vision applications.

RPi is not an ideal platform for real-time systems applications, however, it can be combined with real-time service processors to interconnect real-time microcontrollers to the RPi via electrical buses (e.g., I2C, UART) and Ethernet, this will make the RPi act as the central processor for a distributed control system [10] [11].

III.3.1.b Raspberry PI models

There are mainly 4 categories or Raspberry-PI:

Raspberry Pi Model A: Adapted for low-cost project that needs a complete computer with no networking capabilities and decent I/O support.

Raspberry Pi Model B: This model can be used for a project where price is no object and the most powerful Pi is needed. This model also contains easy-to-use I/O.

Raspberry Pi Compute: This model is best for industrial applications where many I/O lines are needed. This model also maintains strong CPU capabilities.

Raspberry Pi Zero: This model is best for an ultra-low-cost & low-power, tiny-space-constrained project that requires a fully functioning computer and would benefit from wireless connectivity.

Table III.1 present an exhaustive list of different RPi models [12]:

Product	SoC	Speed	RAM	USB Ports	Ethernet	Wireless	Blueto oth
Raspberry Pi Model A+	BCM2835	700MHz	512MB	1	No	No	No
Raspberry Pi Model B+	BCM2835	700MHz	512MB	4	100Base-T	No	No
Raspberry Pi 2 Model B	BCM2836/7	900MHz	1GB	4	100Base-T	No	No
Raspberry Pi 3 Model B	BCM2837A0	1200MHz	1GB	4	100Base-T	802.11n	4.1
Raspberry Pi 3 Model A+	BCM2837B0	1400MHz	512MB	1	No	802.11ac/n	4.2
Raspberry Pi 3 Model B+	BCM2837B0	1400MHz	1GB	4	1000Base-T	802.11ac/n	4.2
Raspberry Pi 4 Model B	BCM2711	1500MHz	2/4/8GB	2xUSB2, 2xUSB3	1000Base-T	802.11ac/n	5.0
Raspberry Pi Zero	BCM2835	1000MHz	512MB	1	No	No	No
Raspberry Pi Zero W	BCM2835	1000MHz	512MB	1	No	802.11n	4.1
Raspberry Pi Zero WH	BCM2835	1000MHz	512MB	1	No	802.11n	4.1

Product	SoC	Speed	RAM	USB Ports	Ethernet	Wireless	Blueto oth
Raspberry Pi 400	BCM2711	1800MHz	4GB	1xUSB2, 2xUSB3	1000Base-T	802.11ac/n	5.0

***Table III.1**: Exhaustive list of different RPi models with their characteristics.*

Figure III.2 Present the evolution of Raspberry-Pi models over the time.

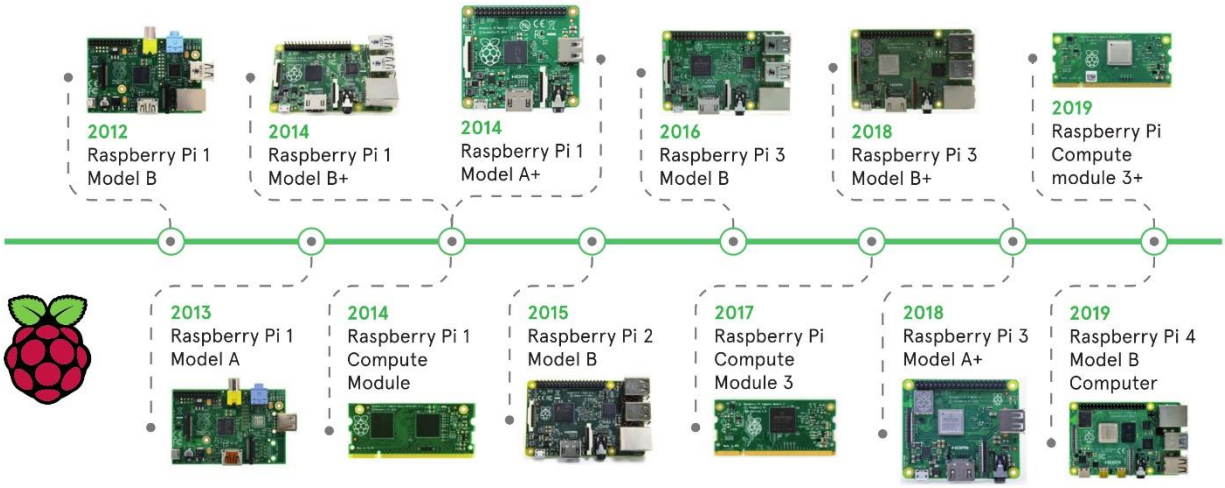

***Figure III.2:** The evolution of Raspberry-Pi models*

In order to design and build a Raspberry Pi project, some application requirements should be taken into consideration, to determine the appropriate board to use. Such requirements can usually be reduced to the following list [13].

- **Speed:** Processing power/performances of the system (CPU, GPU...etc.);
- **Memory:** How much RAM and ROM or storage space the system has;
- **Size and weight:** The physical size and weight of the system;
- **Cost:** The cost of the system;
- **I/O:** How much I/O support is available.

III.3.1.c Raspberry Pi 4 Specifications

Pi 4 Model B is the latest product in the Raspberry Pi range of SBC. It offers more capabilities in terms of processor speed, multimedia performance, memory, and connectivity compared to

the prior-generation Raspberry Pi 3 Model B+, while retaining backwards compatibility and similar power consumption. Figure III.3 shows the Raspberry-PI 4 model B board:

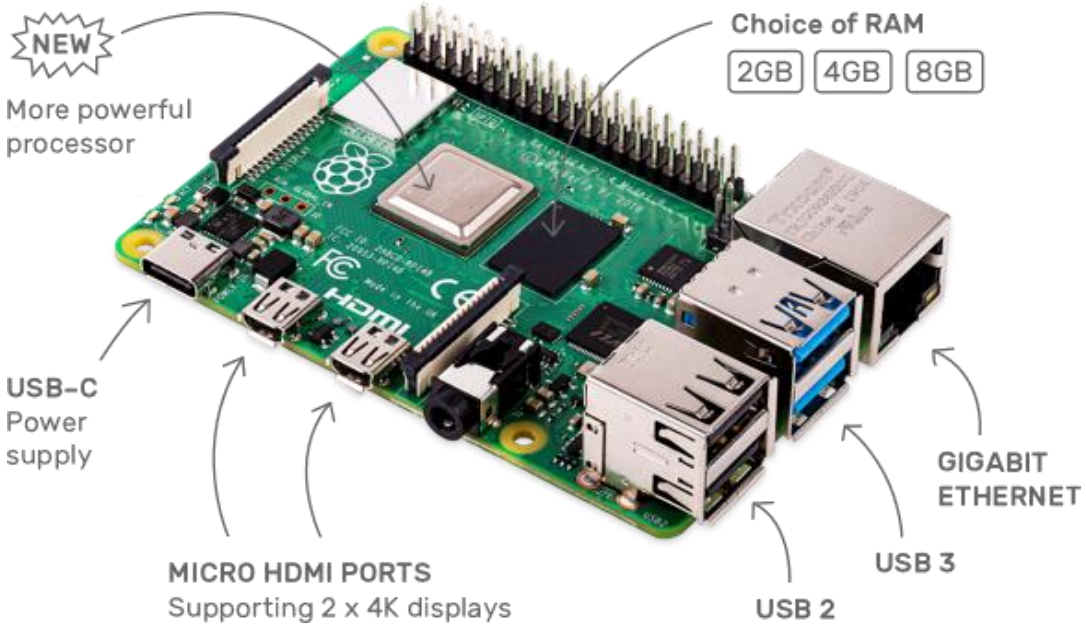

***Figure III.3**: Raspberry-Pi4 Board*

Raspberry Pi 4 comes with several improvements across its hardware components like CPU, GPU, RAM, Networking…etc; witch signifiquently improved the overall performances. The next section will describe the Hardware specifications of RPi4 model B.

- **SoC:** Broadcom BCM2711B0 quad-core A72 (ARMv8-A) 64-bit @ 1.5GHz;
- **GPU:** Broadcom VideoCore VI;
- **Networking:** 2.4 GHz and 5 GHz 802.11b/g/n/ac wireless LAN;
- **RAM:** 1GB, 2GB, 4GB or 8GB LPDDR4-2400 SDRAM;
- **Bluetooth:** Bluetooth 5.0, Bluetooth Low Energy (BLE);
- **GPIO:** 40-pin GPIO header, populated;
- **Storage:** microSD;
- **Ports:** 2 × micro-HDMI 2.0, 3.5 mm analogue audio-video jack, 2 × USB 2.0, 2 × USB 3.0, Gigabit Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI);
- **Dimensions :** 88 mm × 58 mm × 19.5 mm, 46 g.

For the end user, Raspberry Pi 4 Model B provides desktop performance comparable to entry-level x86 PC systems (like i3 old generation or Intel atom).

Since our project is based on Computer Vision, we used the latest version of RPi in our system, the Raspberry Pi 4 boasts a host of impressive specs, from a more powerful processor, to the ability to handle dual 4K displays. It does offer a level of performance that could make it an attractive option for embedded engineers looking to develop consumer-grade IoT products with AI based projects like our case.

The Raspberry Pi uses a variety of input/output devices based on protocols such as HDMI, USB, and Ethernet to communicate with the outside world. In the following list present different interfaces will and protocols used in RPi [14]:

- 802.11 b/g/n/ac Wireless LAN;
- Bluetooth 5.0 with BLE;
- 1x SD Card;
- 2x micro-HDMI ports supporting dual displays up to 4Kp60 resolution;
- 2x USB2 ports;
- 2x USB3 ports;
- 1x Gigabit Ethernet port (supports PoE with add-on PoE HAT);
- 1x Raspberry Pi camera port (2-lane MIPI CSI);
- 1x Raspberry Pi display port (2-lane MIPI DSI);
- 28x user GPIO supporting various interface options:
  - Up to 6x UART;
  - Up to 6x I2C;
  - Up to 5x SPI;
  - 1x SDIO interface;
  - 1x DPI (Parallel RGB Display);
  - 1x PCM;
  - Up to 2x PWM channels;
  - Up to 3x GPCLK outputs.

When it comes to power requirement, The Pi4 needs an USB-C power supply capable of delivering 5V at 3A. If attached downstream USB devices consume less than 500mA, a 5V, 2.5A supply may be used.

GPIO Interface

The Pi4.B makes 28 BCM2711 GPIOs available via a standard Raspberry Pi 40-pin header. This header is backwards compatible with all previous Raspberry Pi boards with a 40-way header as shown in Figure III.4.

As well as being able to be used as straightforward software controlled input and output (with programmable pulls), GPIO pins can be switched (multiplexed) into various other modes backed by dedicated peripheral blocks such as **I2C**, **UART** and **SPI**.

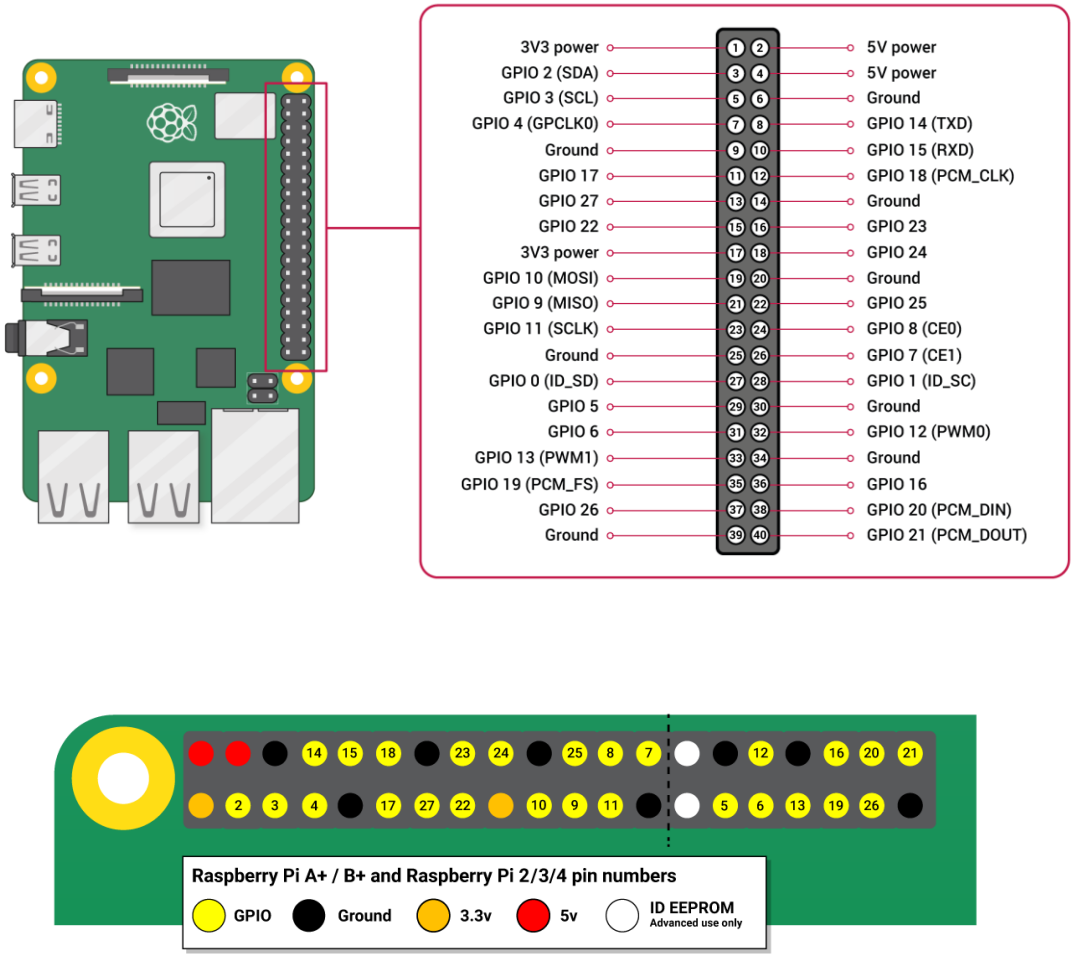

***Figure III.4**: GPIO Connector Pinout*

In addition to the standard peripheral options found on legacy RPi, extra I2C, UART and SPI peripherals have been added to the BCM2711 chip and are available as further mux options on

the Pi4. This gives users much more flexibility when attaching add-on hardware as compared to older models.

Display Parallel Interface (DPI):

An up-to-24-bit parallel RGB interface is available on all Raspberry Pi boards with the 40-way header and the Compute Modules. This interface allows parallel RGB displays to be attached to the Raspberry Pi GPIO.

SD/SDIO Interface:

The Pi4B has a dedicated SD card socket which supports 1.8V, DDR50 mode (at a peak bandwidth of 50 Megabytes / sec). In addition, a legacy SDIO interface is available on the GPIO pins.

Camera and Display Interfaces:

The Pi4B has 1x Raspberry Pi 2-lane MIPI CSI Camera and 1x Raspberry Pi 2-lane MIPI DSI Display connector. These connectors are backwards compatible with legacy Raspberry Pi boards, and support all of the available Raspberry Pi camera and display peripherals.

USB:

The Pi4B has 2x USB2 and 2x USB3 type-A sockets. Downstream USB current is limited to approximately 1.1A in aggregate over the four sockets.

HDMI:

The Pi4B has 2x micro-HDMI ports, both of which support CEC and HDMI 2.0 with resolutions up to 4Kp60.

Audio and Composite (TV Out):

The Pi4B supports near-CD-quality analogue audio output and composite TV-output via a 4-ring TRS'A/V' jack. The analog audio output can drive 32 Ohm headphones directly.

Temperature Range and Thermals:

The recommended ambient operating temperature range is 0 to 50°C. To reduce thermal output when idling or under light load, the Pi4B reduces the CPU clock speed and voltage. During heavier load the speed and voltage (and hence thermal output) are increased. The internal

governor will throttle back both the CPU speed and voltage to make sure the CPU temperature never exceeds 85 degrees C.

Figure III.5 shows that the CPU and controllers are the places where heat accumulate the most, this is why heatsink and fan cooler are mandatory.

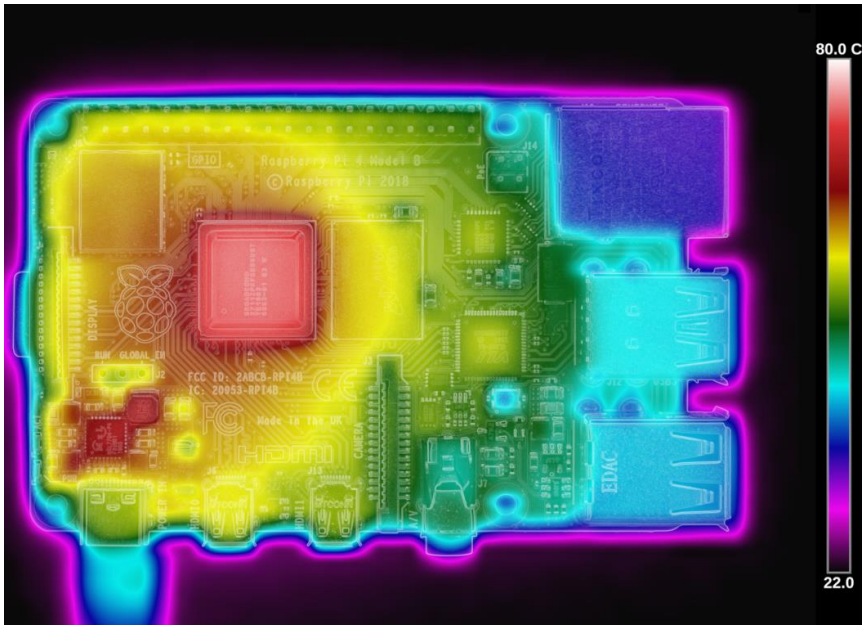

Figure III.5*: Picture from thermal imaging camera and Raspberry Pi 4B*

The Raspberry Pi 4 will operate perfectly without any extra cooling and is designed for sprint performance - expecting a light use case on average and ramping up the CPU speed when needed. However, if a user wishes to load the system continually or operate it at full performance, then heatsinks are really required (presented on Figure III.6), since we deal with high computational power project (Computer Vision and Deep Learning) this cause high CPU & GPU usage which can generate much heat on the surface when it operate at a high temperature at full performance, so, further cooling may be needed (Cooling-Fan can be used to cool down the heatsinks like indicated in Figure III.7).

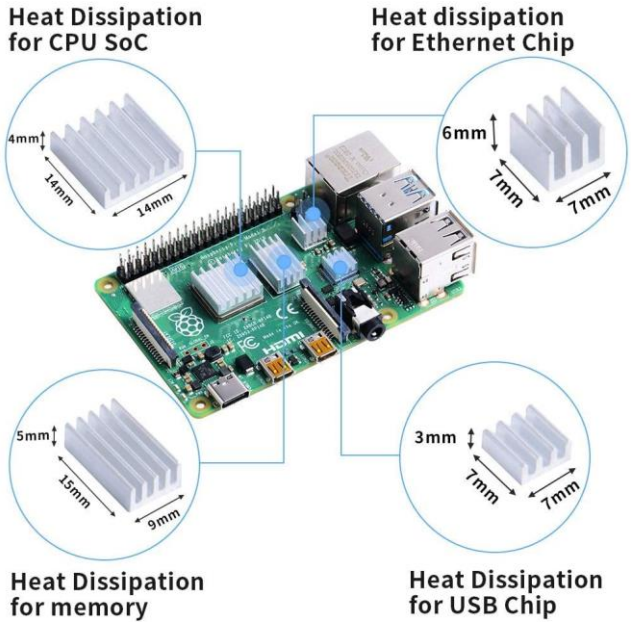

***Figure III.6**: Raspberry-Pi with heatsinks attached to it.*

***Figure III.7**: A Miniature 5V Cooling Fan for raspberry-Pi.*

III.3.2 Raspberry Modules for computer vision

Since our system is a Computer Vision based project, a vision sensor is required, we chose Raspberry Pi Camera Module Rev 1.3 - 5 Megapixel in our project (Figure III.8):

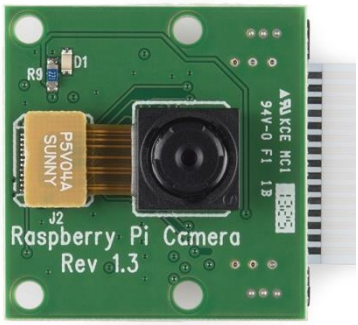

***Figure III.8**: A picture of Raspberry Pi Camera Module Rev 1.3 - 5 Megapixel*

This Camera Module is mandatory for our intelligent surveillance system and it's considered as a custom designed add-on for Raspberry Pi. It attaches to Raspberry Pi by way of one of the two small sockets on the board upper surface. This interface uses the dedicated CSI interface, which was designed especially for interfacing with cameras.

Specification:

The list of specifications and feature of Raspberry Pi Camera Module Rev 1.3:

- Raspberry Pi Camera supports all revisions of the Raspberry Pi;
- 5MP O5647 Camera Module;
- Interface Type: CSI(Camera Serial Interface);
- Still Picture Resolution: 2592 x 1944;
- Video: Supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 Recording.

III.3.3 Raspberry Software development

Raspbian OS has been the most used among the operating systems due to the boost provided by the Raspberry Pi foundation. It is a Debian based Linux operating system optimized for Raspberry Pi boards and it comes with built-in APIs and driver support while maintaining the 32bit & 64bit versions. The following list describes briefly some RPi software specifications:

- ARMv8 Instruction Set;
- Mature Linux software stack;
- Actively developed and maintained;

- Recent Linux kernel support;
- Many drivers up-streamed;
- Stable and well supported Userland;
- Availability of GPU functions using standard APIs.

III.3.3.a The Operating System

To do anything useful with our Raspberry Pi, we need to have an operating system. Raspberry Pi uses the GNU/Linux OS stored on an SD card. Linux is a free and open-source operating system, it's known over the world for its versatility, low power consumption, reliability, security and ease of use. Because it's an open project, many different Linux distributions (or distros) are been made for variety of purposes. Some are made to be feature rich; some are optimized for performances and others are built specially for a category of embedded systems (smartphones, TVs, routers, SBCs…) which are called "Embedded Linux". An embedded Linux refers to a scenario where an embedded system like Raspberry-PI runs on an operating system based on the Linux kernel which will be specifically designed for it.

Linux is ideal for embedded systems because it is flexible, low-cost and open source, and it has already been ported to custom-purpose microprocessors. Compared to proprietary embedded operating systems, Linux allows for multiple suppliers of software, development and support; it has a stable kernel; and it facilitates the ability to read, modify and redistribute the source code. Linux has many supported chip architectures, and so can run on devices as small as sockets and as large as mainframes [13] [11].
The Raspbian OS includes a GUI called the Lightweight X11 Desktop Environment (LXDE) and different programming languages preinstalled like Python and C/C++.

III.3.3.b The Raspberry-Pi Kernel:

The RPi kernel is the main component of a Linux operating system (OS), it's the core interface between a device's hardware and its processes and communicates between the 2, managing resources as efficiently as possible.

The Linux kernel has full control over the device, it adapts to differences in hardware through loadable kernel modules (LKMs) that extend the kernel with device-specific code. LKMs include things like device drivers and file systems.

Kernel Space and User Space:

The Linux kernel runs in an area of system memory called the **kernel space**, and regular user applications run in an area of system memory called **user space**. A hard boundary between these two spaces prevents user applications from accessing memory and resources required by the Linux kernel.

This helps prevent the Linux kernel from crashing due to badly written user code, and because it prevents applications that belong to one user from interfering with applications and resources that belong to another user, it also provides a degree of security.

The Linux kernel has full access to all of the physical memory and resources on the RPi. Therefore, we have to make sure that only the most stable and trusted code is permitted to run in kernel space.

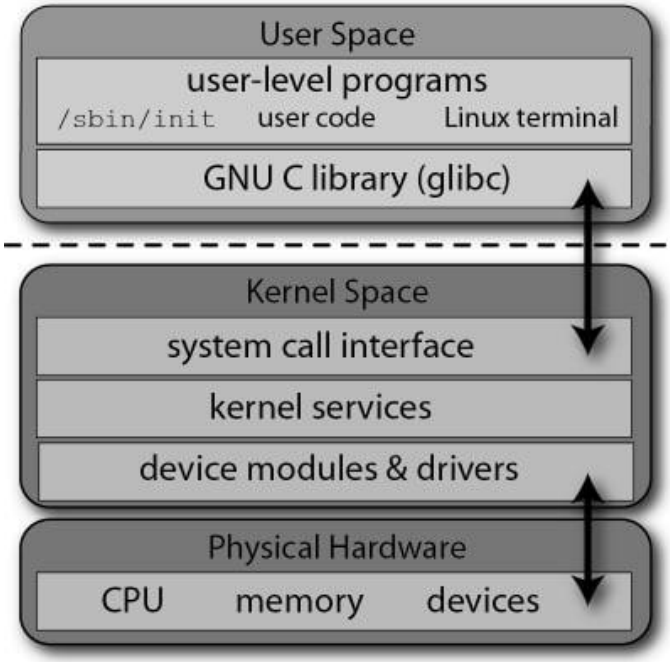

***Figure III.9**: The Linux user and kernel space architectures*

We can see the architectures and interfaces illustrated in Figure III.9, where user applications use the GNU C Library (glibc) to make calls to the kernel's system call interface. The kernel services are then made available to the user space in a controlled way through the use of system calls.

Tweaking the RPi CPU Frequency:

The clock frequency of the RPi was adjusted dynamically at run time. The RPi has various ***governors*** that can be used to profile its **performance**/**power** usage ratio. For example, for a battery-powered RPi application that has low processing requirements, we could reduce the clock frequency to conserve power. But for our case, we clearly need **performance** governor since we deal with high performance project (Computer Vision) [10] [11].

In order to tweak our kernel, we first need to install an extra debian package "cpufrequtils". Then we need to check our current CPU frequency and used GOVERNOR configured in our RPi by typing the command shown in Figure III.10:

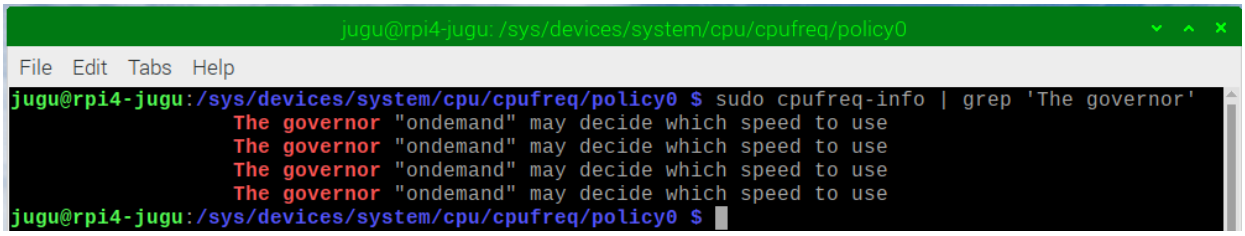

Figure III.10: *Command to display cpufreq governor on Raspberry-Pi*

As we can see the output, the RPi4 has four CPU cores (0–3), the pipeline with command ***grep*** will allow us to search the string "The governor" on each core, we have 4 lines as output, which mean we have 4 cores, each core is configured to be "**ondemand**", The different available cpufreq governors are: **conservative, ondemand, userspace, powersave, performance** and **schedutil**. To enable one of these governors or to explicitly set the clock frequency, we need to enter the commands shown in Figure III.11:

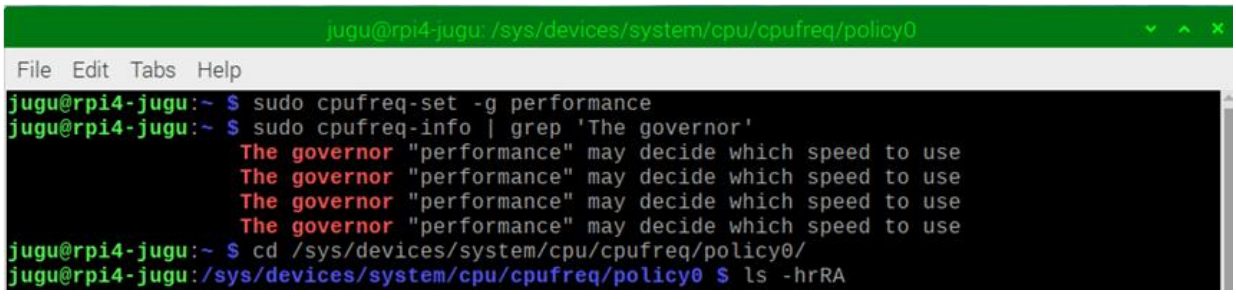

Figure III.11*: Enable a specific cpufreq governor on Raspberry-Pi*

As we can see, after executing the command, the governor is now changed to "**performance**".

Finally, as we mentioned already in **Appendix(H)** and **(I)**, **Linux file system** directories like ***/proc*** and ***/sys*** are just virtual file system, which means they do not actually exist in Hard-Drive, but they are loaded into the RAM on the fly after booting the RPi, since it's in the RAM, they are being deleted after each system reboot (similar to **RAM disk**).
In order to permanently change the default **governor** on the RPi to be **performance** rather than **ondemand** like our case, we need to edit the *cpufrequtils* file *in* ***/etc/init.d/*** (Figure III.12), this will load the tweaked configurations at each boot:

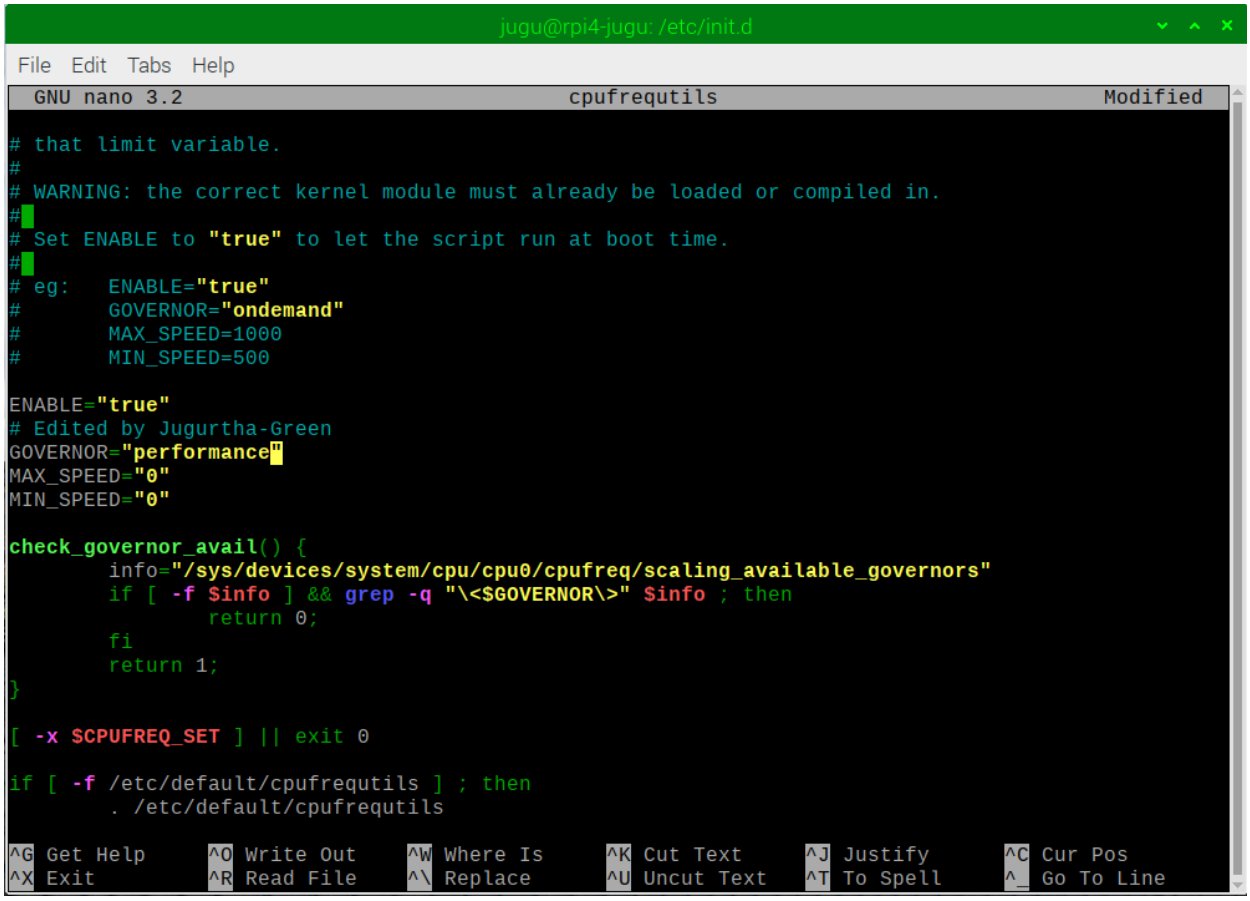

Figure III.12: *Editing the cpufrequtils file to change the governor permanently.*

III.3.3.c Programming on Raspberry Pi:

All programming language that are available under Linux, are also likely to be available for the RPi. However, choosing a suitable programming language depends on what we intend to do with the board. Either interfacing to electronics devices/modules or write a device driver for

Linux where performances are very important. Each of those cases will impact the decision regarding which language to use.

Performance of Languages on the RPi:

If you ask which language performs the best on the RPi, Well, that is an incredibly emotive and difficult question to answer. Different languages perform better on different benchmarks and different tasks. In addition, a program written in a particular language can be optimized for that language to the point that it is barely recognizable as the original code. Nor is speed of execution always an important factor; you may be more concerned with memory usage, the portability of the code, or the ability to quickly apply changes, readability, availability of libraries...etc. However, when it comes to AI, ML and data science. Python is the adopted choice.

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability while supporting several programming paradigms. It usually involves imperative and object-oriented functional programming. It has a comprehensive and large standard library that has automatic memory management and dynamic features.

Python is installed by default on the Raspbian OS image and it is widely used within the RPi community for very good pedagogical reasons, but as users turn their attention to more advanced applications, it is difficult to justify the performance deficit since python is an interpreted programming language. However, we can fix this issue either using the **Cython** Implementation, or combining Python with **C/C++** to improve the performance [11].

III.3.3.d Required Libraries

Before diving deep into our project, we need to afford some software requirements just like the case with the Hardware requirements. The required Python libraries are:

OpenCV-Python

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python **wrappers** that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in

background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation [15].

OpenCV-Python makes use of **Numpy**, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as **SciPy** and **Matplotlib**.

ImageZMQ:

When it comes to live video streaming with OpenCV, there are a *ton* of different options to use. We can go with the IP camera route. But IP cameras can be a pain to work with. Some IP cameras don't even allow us to access the RTSP (Real-time Streaming Protocol) stream. Other IP cameras simply don't work with OpenCV's *cv2.VideoCapture* function. An IP camera may be too expensive as well.

In those cases, we are left with using a standard webcam. The issue then becomes; *how do we stream the frames from a webcam using OpenCV.*

Using FFMPEG or GStreamer is definitely an option. But both of those can be so difficult to work with (especially with people who are out of time like us). After a deep research, we found a solution using *message passing libraries*, specifically *ZMQ* and *ImageZMQ*.

imageZMQ is a set of Python classes that transport OpenCV images from one computer to another using *PyZMQ* messaging [16].

imageZMQ is a transport mechanism for a distributed image processing network. For example, a network of a dozen Raspberry Pis with cameras can send images to a more powerful central computer. The Raspberry Pis perform image capture and image processing like object-detection, blurring and motion detection. Then the images are passed via *imageZMQ* to the central computer for more complex image processing like image tagging, text extraction, feature recognition or Video Summarization.

Features of imageZMQ:

- Sends OpenCV images from one device to another using ZMQ.
- Can send JPEG/PNG compressed OpenCV images, to lighten network loads.
- Uses the powerful ZMQ messaging library through PyZMQ bindings.
- Allows a choice of 2 different ZMQ messaging patterns (REQ/REP or PUB/SUB).

- Enables the image hub to receive and process images from multiple image senders simultaneously.

***The reasons we chose ZMQ and not some other messaging protocol*:**

There are a number of high quality and well maintained messaging protocols for passing messages between computers. We looked at MQTT, RabbitMQ, AMQP and ROS as alternatives. we chose ZMQ and its Python PyZMQ bindings for several reasons:

- ZMQ does not require a message broker. It is a peer to peer protocol that does not need to pass an image first to a message broker and then to the imagehub. This means fewer running processes and less "double handling" of images. OpenCV images are large compared to simple text messages, so the absence of a message broker is important.
- ZMQ is very fast for passing OpenCV images. It enables high throughput between image senders and image hubs.
- ZMQ and its PyZMQ bindings are easy to install.

imageZMQ has been tested for transporting images from a dozen Raspberry Pi computers scattered around to linux image hub servers. The RPi's capture and send dozens to thousands of frames a day. **imageZMQ** has proved to work very reliably and is very fast [16].

NumPy:

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance [17]. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.

- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays.

Beside to python libraries, we also need a model for real-time object detection to detect targets in our captured frames from Client RPi. We chose YOLO (as described in previous chapter) because of its *speed* and *accuracy*, which are both important criteria in our project.

III.4 Conclusion

After presenting the Raspberry-Pi 4 specifications and peripherals, we moved into Software and hardware implementation of our project; this gave us global descriptions of our requirements. In the Hardware implementation section, we have seen the Raspberry-Pi Ecosystem and the different hardware component/modules required to build our system. In the Software implementation section, we introduced the Embedded Linux then we explained the different steps used to Tweak the configurations of our RPi to fit our application. Finally, we had a look about the different python library required for our application to work properly.

In the next chapter, we will explain the system architecture and how it performs, then, we will briefly explain our application code and see its outputs, finally we will close this chapter with some additional features that made this project more useful.

Chapter IV

Intelligent Embedded Vision based MVS System Implementation

Chapter - IV - Intelligent Embedded Vision based MVS System Implementation

IV.1 Introduction

The connectivity of IoT devices creates massive amounts of data. The video data provided by vision sensors in IoT, namely cameras deployed in industries, meets the criteria of Big Data and is quickly becoming the primary sensor device for different IoT applications.

A single vision sensor in an IoT with **30** frames per second (fps) generates huge amount of video data hourly. As IoT is the integration of interconnected smart devices, which shows that there are multiple cameras installed at different places and yields huge amount video data. The amount of cameras exponentially lifts up the amount of video data, making it Big Data repositories. This data requires efficient processing in industries for several purposes such as employee's monitoring and salient events detection, suspicious object detection...etc.

The basic requirements of such data in industries include redundancy removal along with presentation and preservation of only important data in compact form for future use and analysis.

The mainstream devices connected in IoT are resource constrained with limited computation power and storage which cannot process such big video data. Thus, the generated data are possibly transmitted to cloud with unlimited computational power and storage resources for further analysis. Cloud computing is considered as suitable place to analyze such Big Data efficiently. However, the issue with cloud-based solutions is that they are offline and lack reliability. There is always a huge IoT traffic (video data) that should be transmitted over wireless networks in real time. It is obvious that this cannot be guaranteed via cloud computing. Furthermore, along with processing such Big Data, storing it for future use is also a big challenge. It is very difficult in an IoT environment for a resource constrained device to store 108000 frames for a single camera (3600x30fps) and 432000 for a network of four cameras per hour. It requires huge storage devices, however, that is not feasible in an IoT environment.

Thus, keeping only important and representative information of whole day lengthy videos is a better option in terms of limited storage in IoT [4].

IV.2 System Implementation

Our project can be used in industrial environments for various applications such as security and smart transportation and can be proved beneficial for saving resources. In this section, we will present the system Implementation and how it performs.

IV.2.1 Video summarization approaches

In this project, we develop an intelligent IoT based framework with embedded vision for suspicious objects detection, traffic density information sharing, and MVS.

The key features of our system are summarized as follows:

1. MVS processes huge amount of video data generated from distributed video sensors. Majority of existing MVS techniques receive multi-view videos through wireless network and generate offline summary. This requires maximum communication bandwidth and wastes storage capacity. In this project, we present an MVS technique which can generate online summary, compress the keyframes, and transmit them to smart devices connected in IoT network for further analysis. Thus, our system saves communication bandwidth and provides online MVS.
2. It is very hard to install cameras, connect them with computers through wires, and monitor videos manually for targeted objects in industries. Further, the majority of available MVS techniques are very expensive in terms of processing time and hardware implementation which rely only on summary generation. In this framework, we tackle the problem of hardware implementation by installing an embedded device with camera and for suspicious object detection problem we investigate light-weight CNNs for efficiency. The key contribution of our system is installing a single hardware device capable of detecting and reporting about suspicious objects and traffic density to authorities in industrial setup.
3. The huge amount of video data generated by distributed video sensors need CI algorithms for efficient processing. To achieve the goal of dealing Big Data with CI algorithms in IoT precisely, we employ efficient and light-weight CNN to suppress the redundant video data. Our proposed system reads input video (6-fps) and discards the frames with no salient objects. The salient objects depend on the selected Objet Detector model, Thus,

we deal with Big Data in IoT through CI techniques for accurate reduction of data which assist in further steps and decreases time complexity.

4. The currently employed MVS techniques are intractable to be integrated with IoT and many other smart devices. The proposed system can be integrated with IoT and the output can be easily observed via any smart device. If a device is connected to the said wireless network, it can be accessed anytime and anywhere without sitting in a special surveillance monitoring room. So, we contribute to MVS literature by presenting a framework that is adaptable and can be used in any IIoT/IoT environment for summary generation [4].

IV.2.2 System Process

The system process can be divided into 5 different steps like presented in Figure IV.1:

Step 0: An offline step which fine-tunes an existing object detection network for the desired objects like persons, vehicles, masks, suspicious objects (guns, knives etc.).

Step1: IoT setup with client RPi's and smart display devices connected to a wireless network in an industry. The embedded RPi cameras generate video data.

Step 2: It receives the video data, passes each single frame to the trained model which outputs an annotated frame whose objects are analyzed for traffic density to share with connected IoT devices/administration suspicious objects, and if there is any, an alert is generated.

Step 3: The alert received from **step 2** is shared with the concerned departments and smart industries in IoT environment (like police station).

Step 4: It receives (**n**) number of annotated frames that contain dense targets which are encoded and transmitted to the **Master-RPi** in the same network for keyframes selection.

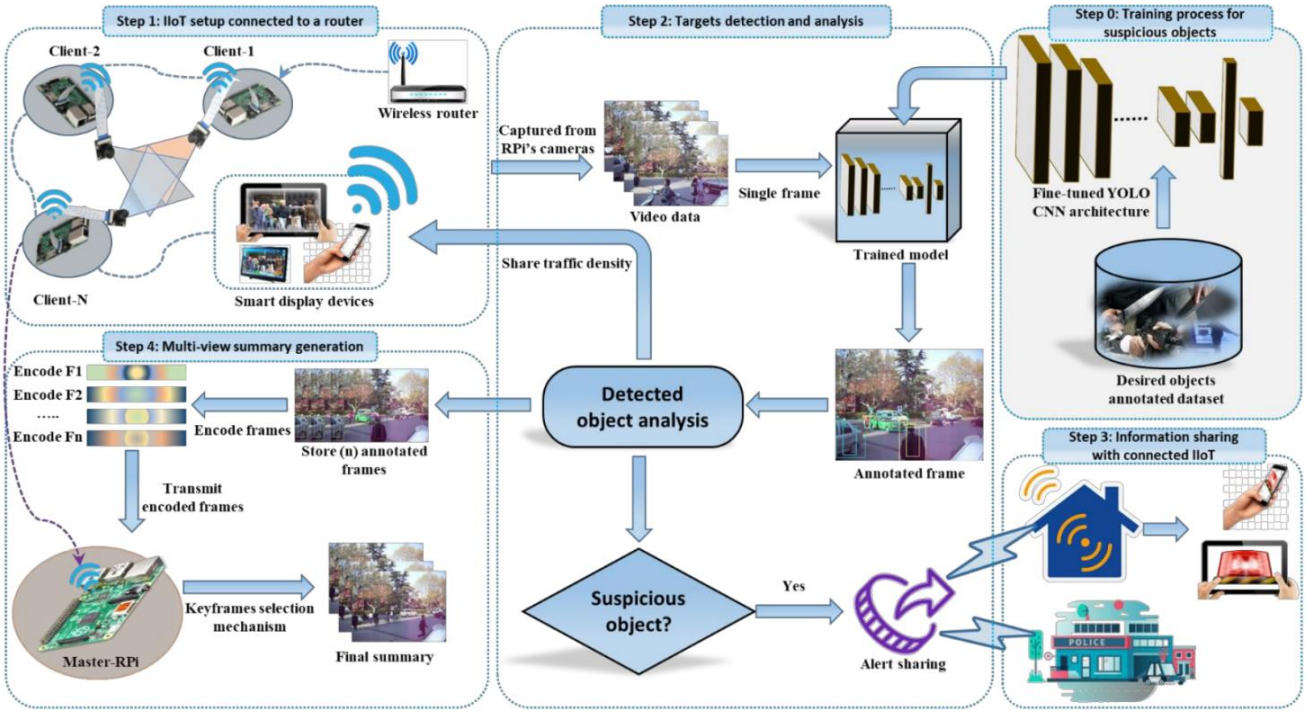

Figure IV.1: *Overall system with different training steps, data acquisition, objects analysis and summary generation*

The descriptions of the main steps are explained bellow; the first step is the offline step, which consist in Object Detection training (using transfer learning):

IV.2.2.a Training Object Detection Model (Offline)

Recently, CNNs showed an outstanding performance for various tasks such as classification, segmentation, retrieval, and object detection. It has been widely used for many applications including action and activity recognition, security and many others. Therefore, we used CNNs for our problem in IoT environment to detect suspicious objects for instant reporting.

The computational complexity of CNNs is a big hurdle to practice CNN based intelligent algorithms over resource constrained devices. To tackle this challenge, we chose a precise, light-weight, and efficient CNN model for object detection. We fine-tune an existing object detection CNN model using transfer learning to detect only *the targeted* objects that can be utilized for summary generation and further analysis. In our project, we can switch between different models of YOLO, we already fine-tuned several models for object detections, some of them are:

- Mask detection (tinyYOLO v4);
- Person & vehicle detection (tinyYOLO v4);
- Hard-Hat detection (tinyYOLO v4);
- Thermal infrared person-dog detection (tinyYOLO v4);
- Person, Cycle, Car, Bike, Bus, Train, Truck detection (tinyYOLO v3).

And the interesting part of our project is that it is programmed in scalable way to easily add new pre-trained models to the list without having to make lot modifications in the code.
We used **YOLO v4 tiny** object detection model in our proposed framework. It is more than 100 times faster compared to a famous objected detection CNN model **Faster-RCNN**. As we used tiny version of YOLO which can process input frames on RPi efficiently. Therefore, we converted the datasets to 6 fps for the input videos so that RPi can process it easily.
The setup process can be seen in the **Appendix(J).**

The dataset acquired is useful in monitoring of smart industries, they are already labelled, but we need to convert the Google CSV file that contains the labelled data into a YOLO data format. Finally, we inputted the images, modified the YOLO configuration file, and pre-trained weights on Google colab (since it provide us free GPU resources) to the training function of YOLO that stores the updated model after every 1000 iterations; we modified the configuration file of tiny YOLO with our desired number of classes and changed the hyper-parameters according to our need, we also added some data augmentation technics like angle, blur, zoom, brightness…etc. this will help us to generalize our model and make robust against environmental effects like lighting, noise, blur and so on.
The next section presents the different models that we have fine-tuned and show their performances:

Thermal-Infrared person-dog detection:

This model contains 2 classes (*dog* and *person* in thermal view), so we changed the number of filters to **21** following the formula **(num_classes + 5)x3** where **num_classes = 2** like show Figure IV.2:

```
num_classes = 2
max_batches = num_classes*2000
steps1 = .8 * max_batches
steps2 = .9 * max_batches
steps_str = str(steps1)+','+str(steps2)
num_filters = (num_classes + 5) * 3
print("writing config for a custom YOLOv4 detector detecting number of classes: " + str(num_classes))
```

Figure IV.2: *Python code shows configuration of number of classes ,max batches and number of filters on YOLO configuration file*

Figure IV.3 display the set of YOLO configurations used to train the model, including several data augmentations (angle, saturation, exposure, HUE, blur…etc.), by default YOLO recommend resolution of 416x416, but we can use any resolution which is a multiple of 32, for example, to get 416, we multiplied 13x32, so we can set any other resolutions like 448, 480, 512, 608 and so on. In this model, 416x416 gave us best result.

```
%%writetemplate ./cfg/custom-yolov4-tiny-detector.cfg
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=24
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.00261
burn_in=1000
max_batches = {max_batches}
policy=steps
steps={steps_str}
scales=.1,.1
```

Figure IV.3 *: Python code shows writing YOLO configurations used to train the thermal infrared person model*

We used the default learning rate recommended by YOLO community (0.00261), with a burn-in on each 1000 iteration, this will let us select the best model at the end according the Mean Average precision (mAP) of the burned model.

Figure IV.4 shows the learning curve of the Thermal-Infrared person-dog model using darknet on the Google Colab. This curve has been automatically generated by the darknet framework, it presents the mAP according to the number of iterations.

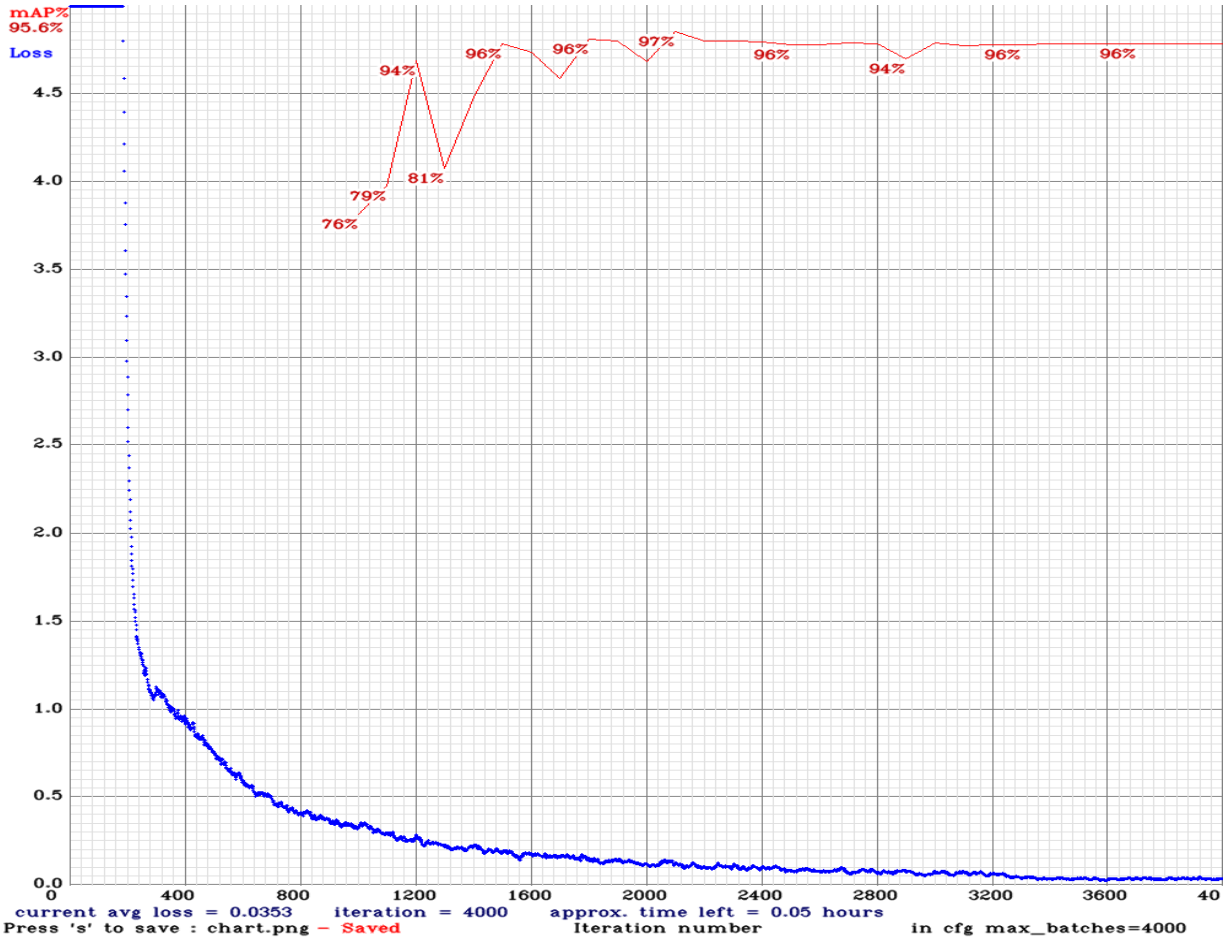

Figure IV.4: *Learning curve for the Thermal-Infrared person-dog model using darknet framwork*

After training step has been completed, the model has reached a mAP of 95.60% with 77.46% of **IoU** (Intersection over Union). We used of dataset of 203 images from ***public.roboflow.com*** which is publicly available.

Performance of our model is shown in Figure IV.5 and seems to be satisfying, however the number of training images is not enough, so the model might not be as performant as it seems:

```
 (next mAP calculation at 4000 iterations)
 Last accuracy mAP@0.5 = 95.60 %, best = 96.93 %
 4000: 0.038486, 0.035271 avg loss, 0.000026 rate, 1.755752 seconds, 192000 images, 0.050202 hours left

 calculation mAP (mean average precision)...
44
 detections_count = 75, unique_truth_count = 49
class_id = 0, name = dog, ap = 95.45%           (TP = 22, FP = 0)
class_id = 1, name = person, ap = 95.75%        (TP = 26, FP = 5)

 for conf_thresh = 0.25, precision = 0.91, recall = 0.98, F1-score = 0.94
 for conf_thresh = 0.25, TP = 48, FP = 5, FN = 1, average IoU = 77.46 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.956011, or 95.60 %
Total Detection Time: 1 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.956011
Saving weights to backup//custom-yolov4-tiny-detector_4000.weights
Saving weights to backup//custom-yolov4-tiny-detector_last.weights
Saving weights to backup//custom-yolov4-tiny-detector_final.weights
```

***Figure IV.5**: Performances of the Thermal Infrared person-dog model on darknet*

Where **TP** stands for True Positive and **FP** for False Positive, the **TP** is an outcome where the model correctly predicts the positive class. Similarly, a True Negative (**TN**) is an outcome where the model correctly predicts the negative class.

The **FP** is an outcome where the model incorrectly predicts the positive class. And a False Negative (**FN**) is an outcome where the model incorrectly predicts the negative class.

Mask detection model:

In this model, we have 2 classes (mask & no-mask), so we changed the number of filters according to the formula **(num_classes + 5) x 3** witch gives us **21.**

```
%%writetemplate ./cfg/custom-yolov4-tiny-detector.cfg
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=24
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0.3
saturation = 1.7
exposure = 1.7
hue=.1
blur=0.3

learning_rate=0.0015
burn_in=1000
```

***Figure IV.6:** Python code shows writing YOLO configurations used to train the mask model*

The set of YOLO configurations used to train this model are shown in Figure IV.6.

Those hyper-parameters have been selected as best choice after several trainings and tests (more than 7 trainings) for this model, this is due to the leak of the dataset which contains only 149 images, and not just that, the dataset contains mostly Chinese people, which made the model less generalist, that's what lead us to use more data augmentation parameters and higher resolution (608 x 608).

Figure IV.7 shows the learning curve of the mask detection model using darknet on the Google Colab.

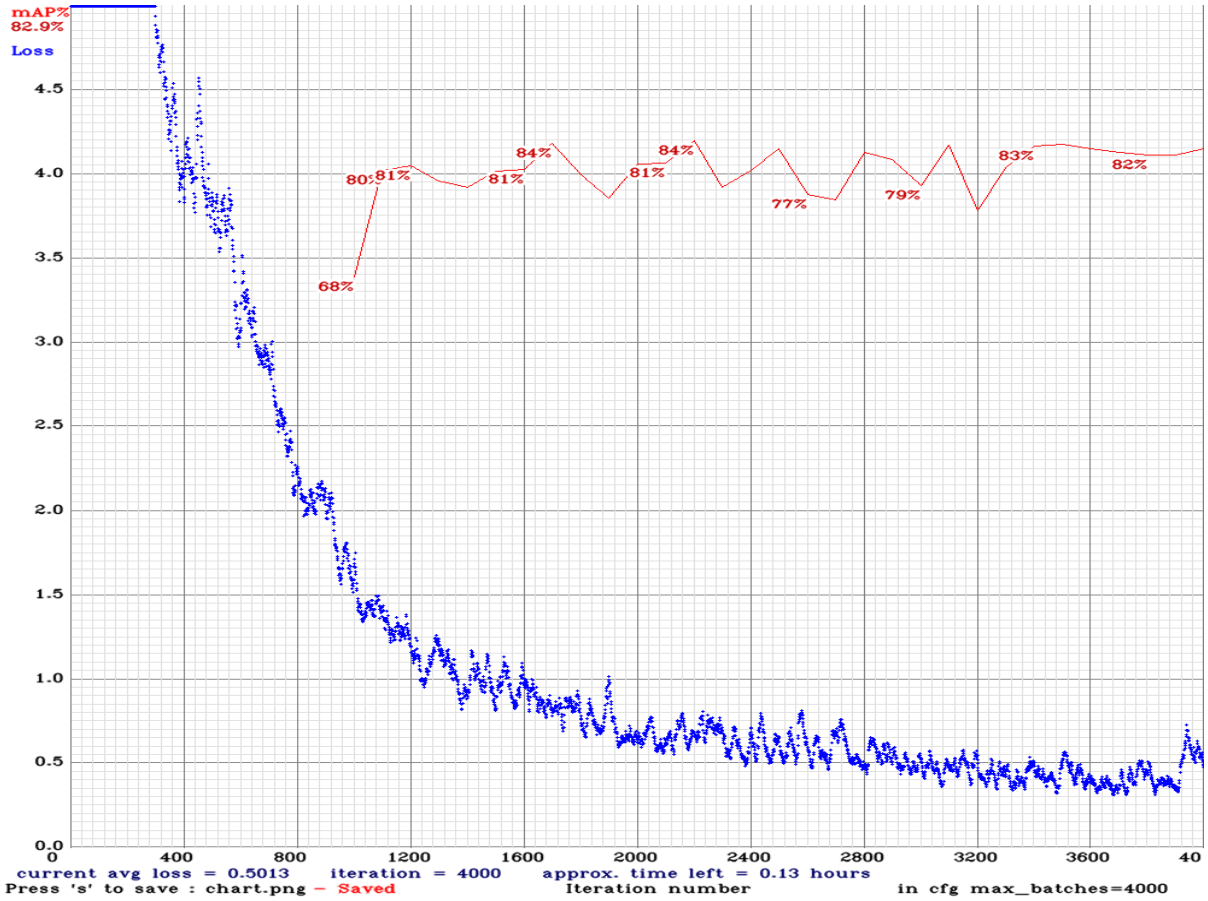

***Figure IV.7**: Learning curve for the mask detection model using darknet framwork*

The model has reached a **mAP** of 82.93% with 71.16% of **IoU**, 87.53% of AP (Average precision) on class "mask" and 78.34% on class "no-mask", and this is due to the fact that there is a slight difference between number of mask images and no-mask images in training dataset. We used a public dataset of 149 images from ***public.roboflow.com***.

The Performances of our model is shown in Figure IV.8, this model might seem not perfect, but it gives good result in real life implementation.

```
 calculation mAP (mean average precision)...
32
 detections_count = 329, unique_truth_count = 162
class_id = 0, name = mask, ap = 87.53%           (TP = 115, FP = 11)
class_id = 1, name = no-mask, ap = 78.34%        (TP = 14, FP = 6)

 for conf_thresh = 0.25, precision = 0.88, recall = 0.80, F1-score = 0.84
 for conf_thresh = 0.25, TP = 129, FP = 17, FN = 33, average IoU = 71.16 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.829337, or 82.93 %
Total Detection Time: 1 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.829337
Saving weights to backup//custom-yolov4-tiny-detector_4000.weights
Saving weights to backup//custom-yolov4-tiny-detector_last.weights
Saving weights to backup//custom-yolov4-tiny-detector_final.weights
```

***Figure IV.8**: Performances of the mask detection model on darknet*

Hard-Hat detection model:

In this model, we have 3 classes (head, helmet, person), so we again changed the number of filters according to the formula **(num_classes + 5) x3** which gives us **24.**

The set of YOLO configurations used to train this model are shown in Figure IV.9.

```
%%writetemplate ./cfg/custom-yolov4-tiny-detector.cfg
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=32
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0.2
saturation = 1.6
exposure = 1.6
hue=.2
blur=0.4

learning_rate=0.00261
burn_in=1000
```

***Figure IV.9:** Python code shows writing YOLO configurations used to train the hard-hat model,*

Those hyper-parameters have been tested and seem to perform better for this model, we used the standard learning rate of 0.00261 and increased a little bit the number of subdivisions with a slight change on data augmentation parameters.

Figure IV.10 shows the learning curve of the hard-hat detection model using darknet on the Google Colab.

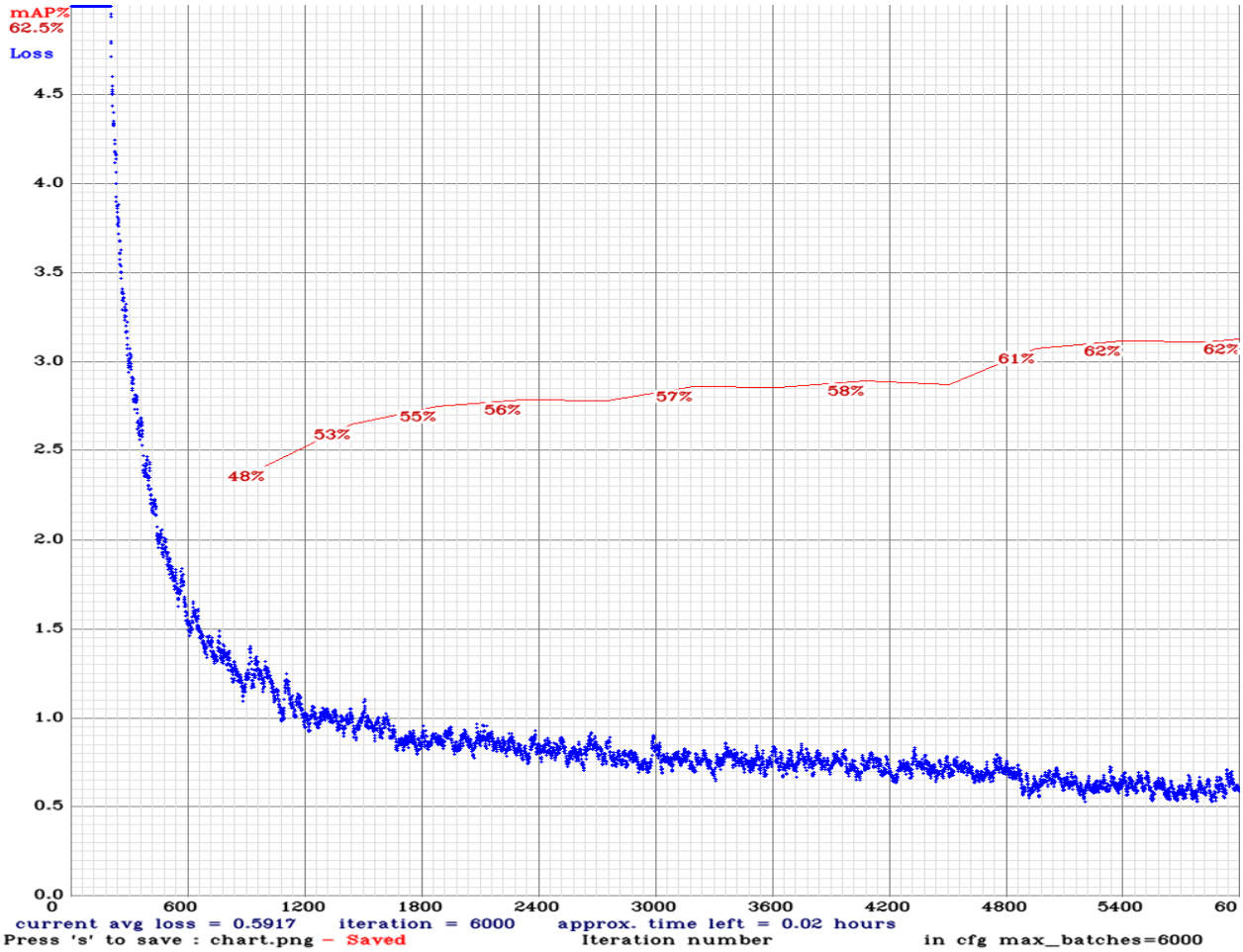

***Figure IV.10**: Learning curve for the hard-hat detection model using darknet framwork*

This model is practically the most precise one among the 3 models we had trained, even though it has only **62.47%** of **mAP** with 76% of **IoU**. This may be strange, but I will explain to you the reason, the dataset has 7041 images which is huge comparatively to the other models, however it's an imbalanced dataset, which means there is much differences between set of images between classes, in our case, **head** and **helmet** classes have enough training images, but the **person** class has only few images, that's why after training we have got:

- Class *head* has an AP of 84.02% with 4490/309 positive detections;
- Class *helmet* has an AP of 85.95% with 16354/1231 positive detections;
- Class *person* has an AP of 15.45 % with 39/20 positive detections.

This mean the model will not detect *persons* correctly but it will detect the *heads* and *helmets* perfectly. And since we only care about hard-hat detection, we don't really need our model to detect *person*, because if it detected a *head* or *helmet* it means it's already a person.

We thought of filtering the dataset manually by removing *person* images, but for 7000 images it's not really practical, beside that, we didn't download the dataset in our local machine, instead, we transferred it directly to *Google Colab* since it's a huge dataset.

The overall Performances of this model is shown in Figure IV.11:

```
 calculation mAP (mean average precision)...
5272
 detections_count = 73304, unique_truth_count = 26852
class_id = 0, name = head, ap = 84.02%           (TP = 4490, FP = 309)
class_id = 1, name = helmet, ap = 87.95%         (TP = 16354, FP = 1231)
class_id = 2, name = person, ap = 15.45%         (TP = 39, FP = 20)

 for conf_thresh = 0.25, precision = 0.93, recall = 0.78, F1-score = 0.85
 for conf_thresh = 0.25, TP = 20883, FP = 1560, FN = 5969, average IoU = 75.93 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.624732, or 62.47 %
Total Detection Time: 32 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.624732
New best mAP!
Saving weights to backup//custom-yolov4-tiny-detector_best.weights
Saving weights to backup//custom-yolov4-tiny-detector_6000.weights
Saving weights to backup//custom-yolov4-tiny-detector_last.weights
Saving weights to backup//custom-yolov4-tiny-detector_final.weights
```

***Figure IV.11**: Performances of the mask detection model on darknet*

IV.2.2.b IoT Setup Connected to a Router

Normally in industries video data from vision sensors is transmitted to cloud for better understanding and efficient analysis, requiring uploading data to cloud. The data transmission to cloud yield huge wastage of communication bandwidth, time, and makes real-time response impossible corresponding to abnormal actions or activities. To overcome these challenges, we propose a novel resource constrained RPi based system. Vision sensor is attached with each RPi and is connected to a wireless network in IoT environment for efficient and intelligent processing of video data.

The client RPis can be installed at locations where multi-view data are important to be captured in smart industries. There can be a network of (n) number of RPi's inter-connected to capture video data individually. Each client RPi captures video data at 6-fps, to make the process online and efficient. The client RPi then passes a single frame to the object detection model that annotates it for the specified targets. The detected objects are analyzed, compressed and encrypted then sent the Master Raspberry-Pi for farther process.

IV.2.2.c Targets Detection and Analysis

The input frames acquired from sub-section above are processed in this step. The trained model in the initial step of our framework is used to annotate frame for configured targets like: vehicles, persons, suspicious objects...etc.

The density of targeted objects (vehicles, persons...) are computed from the annotated frame and shared with the smart devices connected to the IoT network. Information about traffic in smart industries helps building up a routine and plays a key role in saving time. If the annotated frame contains any type of suspicious object such as gun or knife it is considered as an alert.

The alert is shared with the concerned devices in IoT for employee's safety and also with the police department for quick preventive actions. The police department can analyze the situation of alert by watching the camera and if it is not alarming, they can ignore it. After a continuous detection of suspicious object, an alert is sent again to the police department.

IV.2.2.d Multi-view Summary Generation

The final summary is generated on master RPi. The input to this step is (n) number of frames from each Client RPi. We select *n* to be 20. Therefore, we encode 20 frames from each view having many targets like vehicles and persons. We apply lossless PNG compression on these frames and send it to the master RPi. The PNG compression has the advantage of saving communication bandwidth without losing the quality of the image. The master RPi receives PNG compressed frames in the form of a vector and decodes it to restore the original frames then process it using OpenCV-python library.

The same process is applied for rest of the frames and after decoding the frames are processed via two methods: **entropy** and **complexity** to compute the information present in frames.

Entropy:

Image entropy indicates the amount of information inside a frame. The higher value of entropy represents that the frame is rich of information and lower-value of entropy shows that the frame has less amount of information. The process flow of entropy value computation is given in Figure IV.12(**a**).

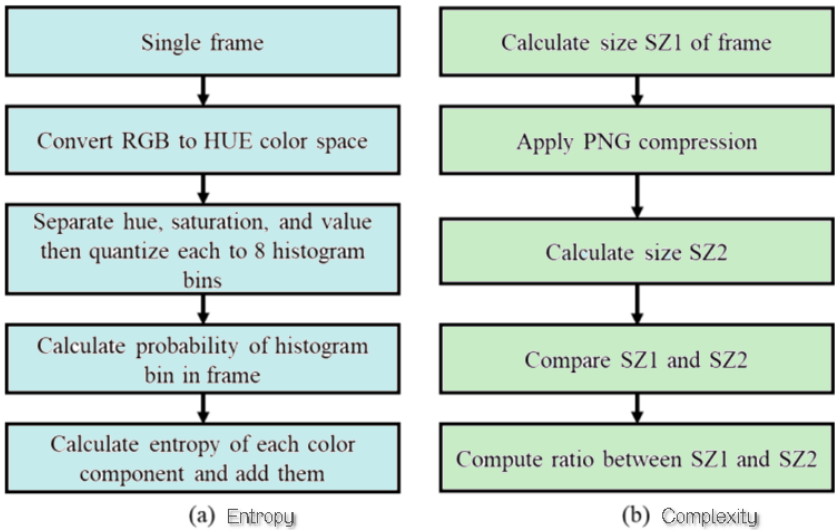

Figure IV.12: *Computing entropy (a) and complexity (b) score from a single input frame*

Complexity:

The complexity of a frame is computed by comparing original size of the frame and its size after PNG compression [36] is applied. The PNG compression algorithm is applied according to the human visual system as explained in. High compression rate shows that the frame has little visually important information and vice versa. Complexity value computation is visualized in Figure IV.12 (**b**).

These two methods output a real value between **0** and **1**, and the frames with highest value of sum for complexity and entropy is selected as a keyframe.

The number of keyframes in single chunk of 20 frames depends upon the configured threshold. For example, if we select single frame preview from the frames extracted from Road dataset in a single chunk. The total frames in a single chunk are (n * number of client RPi's), where n is the number of frames each client RPi's transmits [4].

Figure IV.13 presents the values of entropy and complexity corresponding to the frames having high and low information with and without compression. The results are same for the selection of keyframes indicating no effect of the compression scheme applied over the frames.

***Figure IV.13**: Different Keyframes samples taken from different views*

Frame	Entropy value	Complexity value	Sum	View
150	0.2997	0.5100	0.8097	2
198	0.2986	0.5002	0.7988	2
1224	0.3013	0.5476	0.8489	3
1236	0.2811	0.5044	0.7855	1
Decoded frames (after applying PNG compression)				
150	0.2987	0.5002	0.7989	2
198	0.2975	0.4970	0.7945	2
1224	0.2993	0.5459	0.8452	3
1236	0.2833	0.5035	0.7868	1

***Table IV.1**: Sample video frames from Road dataset videos of different views*

Frame #150: higher entropy and complexity values show high amount of information present in frame and the objects detected are near to the camera which means they are important and need to be considered for the final summary.

Frame #198: the complexity and entropy values are lower because of low amount of information and objects are far from the camera.

Frame #1224: The objects are nearer to the camera and frame contains higher information.

Frame #1236: Although there are many vehicles in this frame but they are far from camera so the information noticed is comparatively lower than Frame #1224.

The keyframes before and after applying encoding/compression scheme are the same. The sum of entropy and complexity for different frames is given in the table where for both the scenarios (before and after compression and decoding) the two keyframes among the four are the same (Frame #150 and Frame #1224). Frames with highest sum are made bold and the second highest sum is made italic and underlined inside the table in Table IV.1.

IV.3 Application code description

In this section, we will explain the general structure of our project, we used the Oriented Object Programming (**OOP**) paradigm with Python 3.9 to make our code more scalable and easy to maintain using the modular architecture.

IV.3.1 Development Environments

We used PyCharm as our main IDE (Integrated Development Environment) during our development because it's recommended by most of the professional developers and it has been considered the best IDE for python developers.

We also used VS Code to code directly from the Raspberry-PI since it supports ARM (Advanced RISC Machine) architecture.

We implemented the ***virtualenv*** tool to create a project-specific isolated virtual environment. The main purpose of virtual environments is to manage settings and dependencies of a particular project regardless of other Python projects.

IV.3.2 Code explanation:

In this section we will explain how python scripts perform, Figure IV.14 gives a general schema on how our application performs on both Client and Master Raspberry-Pi. The application structure can be seen in the **Appendix(K).**

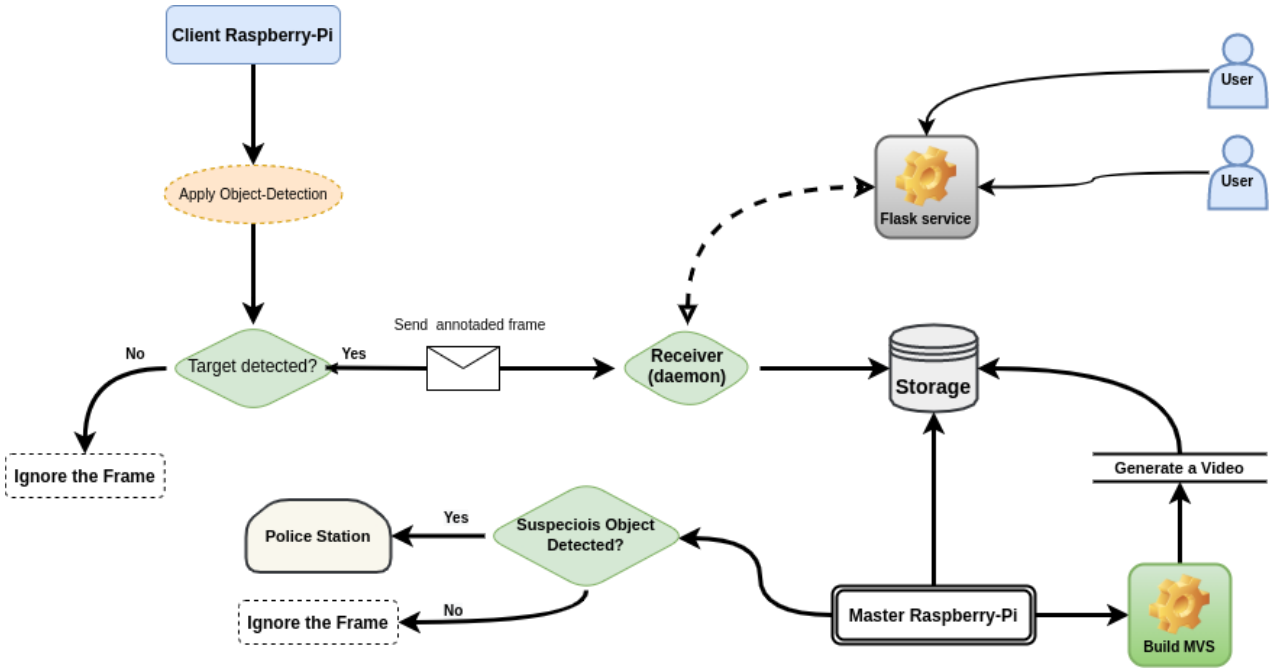

Figure IV.14*: A presentation of our code operation*

IV.3.2.a On Raspberry-Pi Client:

On the Client Raspberry-Pi, the *main_Client_RPi_jugu.py* script will attempt to detect the specified targets (according the chosen model) using the Python class which is inside *object_detector.py* called *Object_Detector* like show Figure IV.15:

```
80          try:
81              image, confidences, class_ids = Object_Detector(self.frame_i, model_type='mask-detection').get_marked_img()
82          except:
83              print("Video Termination.")
84              self.release_resources()
85              time.sleep(5)
86              continue
```

Figure IV.15*: Python code shows instantiation of Object_Detector class*

If a target is detected, then we apply a lossless PNG compression using the **encode_image** method from the *CoDec* class (Figure IV.16), then we send it to the Master RPi**.**

```
276             ##########################################################################
277             path = '' + str(self.shot_number) + '/frame_' + str(self.i) + '_'+self.view_no+'.png'
278             ##########################################################################
279             # cv2.imwrite(path, image) # we don't need to write anything on local storage  since we will
280             # send all the frames to Master-RPi for MVS analysis
281             print(str(self.frames_count)+' - Frame generated : ', path)
282             embedded_msg= self.rpi_name+'@'+path
283             self.sender.send_image(embedded_msg, CoDec.encode_image(img=image) )
284             ##cv2.imshow('f',image)
285             # cv2.waitKey(30)
286
```

Figure IV.16*: Python code shows compressing frames before sending them to Master RPi*

The Master RPi socket is set while creating the instance of Client_RPi class, like show Figure IV.17 (we used port 5555 by default):

```
56          self.sender = imagezmq.ImageSender(connect_to='tcp://anonymous.local:5555') #anonymous is the hostname of my PC.
57          self.rpi_name = socket.gethostname() # the current Client Rpi hostname
```

Figure IV.17*: Python code shows sending frames to Master RPi using ImageZMQ library*

We also initialized the default resolution of the camera and the FPS (in our case 640x480 with 6fps), Figure IV.18 shows the initialization process inside the constructor.

```
27  def __init__(self, video_source=0,fps=6,view_no='v1'):
28      # Initialize property
29      print('Initializing Video input...')
30      self.fps=fps
31      self.video_source=video_source
32      #self.video_source='/home/jugu/Videos/4K Video Downloader/Test kamery HIKVISION DS-2CD2055FWD-I 2.8mm.mp4'
33      ##########################################################################################################
34      self.capture = cv2.VideoCapture(self.video_source)
35      self.capture .set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'))  # depends on fourcc available camera
36      self.capture .set(cv2.CAP_PROP_FRAME_WIDTH, 640)
37      self.capture .set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
38      self.capture .set(cv2.CAP_PROP_FPS, 30)
39      ##########################################################################################################
```

Figure IV.18*: Python code shows configuration of OpenCV instance*

If we check the selected frames according to the used object detection model before sending it to Master raspberry-PI, we will see different results on Figure IV.19, Figure IV.20, Figure IV.21 and Figure IV.22.

The system will send a frame to Master Raspberry-Pi only and only if the targeted object has been detected (according to the selected model) and bypass a certain threshold (entropy + complexity).

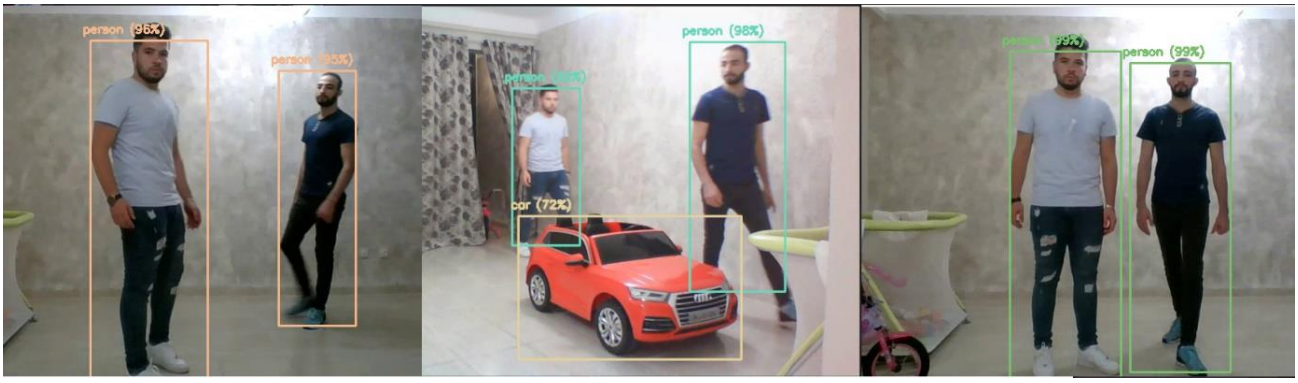

Figure IV.19: *Examples of detected frames on Raspberry-Pi Client while using person-vehicule model*

Figure IV.20: *Examples of detected frames on Raspberry-Pi Client while using hard-hat detection model*

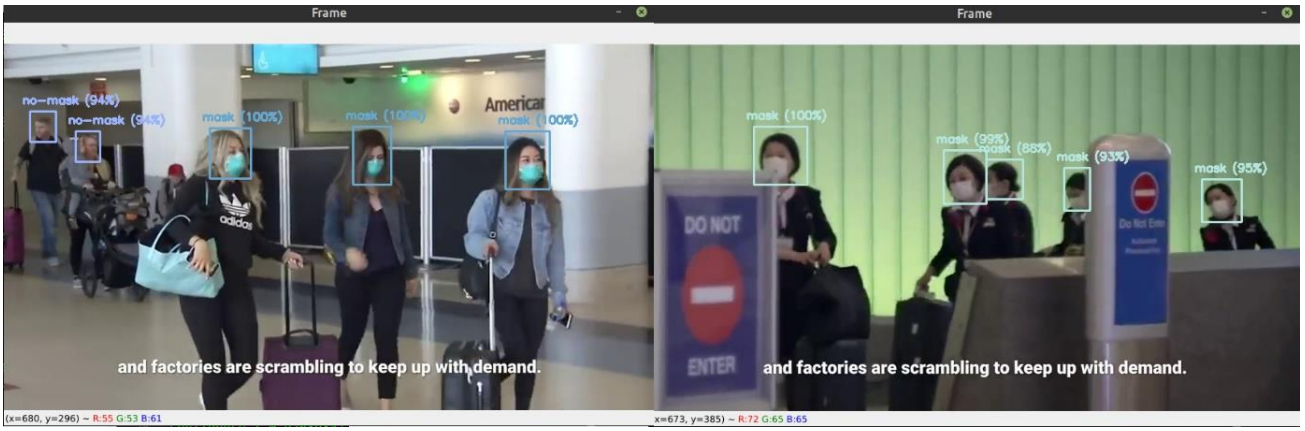

Figure IV.21: *Examples of detected frames on Raspberry-Pi Client while using mask-detection model*

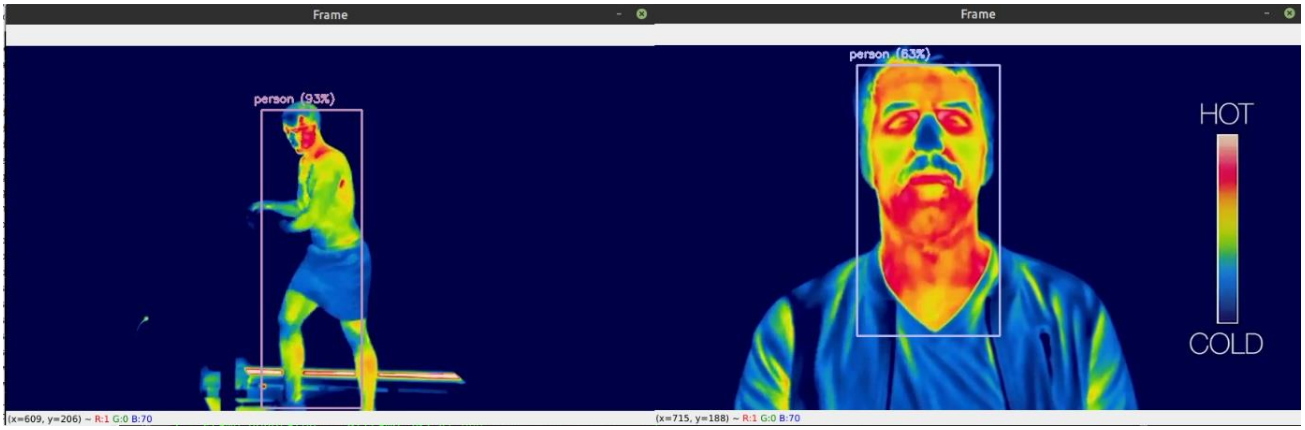

Figure IV.22*: Examples of detected frames on Raspberry-Pi Client while using thermal infrared person detection model*

IV.3.2.b On Raspberry-Pi Master:

On the Master Raspberry-Pi, the script *receiver_service.py* is the daemon process responsible of receiving the compressed images from different Client RPi and store them as keyframes for MVS process.

The Master RPi is continuously listening on port 5555 as shown in Figure IV.23. Ports that are below 1024 will need root access since they are well-known port.

```
123    def __init__(self, open_port='tcp://*:5555', request_response=True):
124
125        self.request_response=request_response
126        # listening on 0.0.0.0:5555 (you may need to allow the port on your machine in your firewall like IPtables in our case
127        self.image_hub = imagezmq.ImageHub(open_port=open_port, REQ_REP=request_response)
```

***Figure IV.23**: Python code shows Master RPi listning to port 5555 for incoming frames*

The script uses Multi-Threading technique to ensure better Clients RPi management; it will try to create a new thread for each new connection, Figure IV.24 explain the process.

```
148    #@property
149    def start(self ):
150        print("starting the service...",end="\n")
151        while True:
152            # local_number_of_connections=0
153            print("Waiting for Clients Raspberry-PI connections...",end="\n")
154            # create a new thread for every connection
155            handler_thread = th.Thread(target=self.handle_connection , args=(self.request_response,))
156            handler_thread.start()
```

***Figure IV.24**: Python code shows using multithreading for each client connection*

The *handle_connection* method is executed on separate thread and it responsible of receiving the encoded image, decode it, the store it as a PNG image (Figure IV.25).

This method is also responsible for decoding the received frames from client Raspberry-Pi (Line 338), Using Dead-Lock algorithm, we will make sure that the out frames are not accidentally being read by other processes/threads while trying to update it.

If enabled in configuration, this method will attempt to send an “OK” message to the sender on each frame to ensure reliability (just like TCP acknowledgment).

```
330     def handle_connection(self, send_OK):
331         global lock
332         global outputFrame
333         while True:
334             embedded_msg, encoded_image = self.image_hub.recv_image()
335             rpi_name,path = str(embedded_msg).split('@')
336             path= ReceiverService.base_path+""+path
337             print(f'Frame received from :{rpi_name} - stored in :  {path}')
338             image = CoDec.decode_image(encoded_image)
339             #outputFrame = image
340             with lock:
341                 outputFrame = image.copy() # acquire the lock, set the output frame, and release the lock
342                 #We need to acquire the lock to ensure the outputFrame variable is not accidentally being read by a client
343                 # while we are trying to update it.
344             #self.generate()
345 
346             if not os.path.exists(os.path.dirname(path)):
347                 print('Creating directory....') # create a directory if not existed
348                 os.makedirs(os.path.dirname(path))
349             #ReceiverService.connected_clients.append(rpi_name)
350             #ReceiverService.number_of_connections+=1
351             #cv2.imshow(rpi_name, image)  # 1 window for each RPi
352             cv2.imwrite(path, image)
353             cv2.waitKey(1)
354             if send_OK: self.image_hub.send_reply(b'OK') # send an OK for each received frame (this will increase reliability)
```

Figure IV.25*: Python code shows handle_connection function*

While the receiver daemon is busy storing the received frames, the script *main_Master_RPi_jugu.py* is processing the Multi-View Summarization step on the stored images, if no image received, it waits until new images are stored as indicated in line 103 of Figure IV.26 by just rising an exception and return **-1**, the output will be interpreted by the global script, if code error -1 is returned from the *calculate_ComplexityEntropy()* function, then it will wait for 2 seconds before next retry. (this will also avoid high CPU stress).

On each image, the script will try performing calculations of Complexity and Entropy, if the summation between those two reaches a certain threshold, then the frame is considered salient.

```
98      # This method is void, it will build a list of Complexity+Entropy values.
99      def calculate_ComplexityEntropy(self, root_path):
100         try:
101             listed_dir= os.listdir(root_path)
102         except:
103             print(f"Waiting for : {root_path} availability...")
104             return -1
105         for self.single_image in listed_dir:
106             print('Processing...' + root_path + self.single_image)
107             image = cv2.imread(root_path + self.single_image)
108             self.image_list.append(self.single_image)
109             entropy_value = Entropy(image).get_entropy  # calculate Entropy value
110             complexity_value = Complexity(image).get_complexity # calculate Complexity value
111             sum_values = entropy_value + complexity_value # Calculate the sum Complexity+Entropy
112             self.all_sum.append(sum_values)
113             print(str(self.counter), 'Entropy score = ', str(entropy_value), ', complexity score = ', str(complexity_value))
114             self.counter +=1
```

Figure IV.26: *Python code shows calculations of entropy and complexity.*

The salient frame is then added to the queue list to build a finale highly compressed video, we can use any video Format supported by OpenCV (it support more than 100 video formats), the exhaustive list of supported video formats is available on www.fourcc.org/codecs.php. In our case we chose AVI (Audio Video Interleave) with 5 fps as shown in Line 175 on Figure IV.27. The *VideoWriter* will try to automatically save the generated video file in case of unexpected error, this makes our system fault tolerant.

```
169             img = cv2.imread(self.root_path+self.single_image)
170             height, width, layers = img.shape
171             size = (width,height)
172             if not already_initialised:
173                 now_utc = datetime.now(timezone.utc)
174                 # Python: cv2.VideoWriter([filename, fourcc, fps, frameSize[, isColor]]) → <VideoWriter object>
175                 self.generated_video = cv2.VideoWriter(self.base_path+'/Generated_Videos_'+str(now_utc)+'.avi',\
176                                                        cv2.VideoWriter_fourcc(*'DIVX'), 5, size)
177                 already_initialised=1
178 
179 
180             print ('Processed.. ' + str(self.dir_counter))
181             self.dir_counter+=1
182             if cv2.waitKey(5) & 0xFF == ord('q'):
183                 break
184             if not already_initialised:continue
185             try:
186                 self.generated_video.write(img)
187             except:
188                 print("Video writer stopped unexpectedly !")
189                 print("Generating the Video File...")
190                 self.generated_video.release()
191                 already_initialised=0
```

***Figure IV.27**: Python code shows generating a video files from received frames.*

After the video is written in the storage, we can configure the system to automatically remove the used keyframes in the videos since we can easily generate again those keyframes from the stored video.

We tested the system with a video of 2 minutes, and at the end the MVS process, the system has generated us the following files (Figure IV.28):

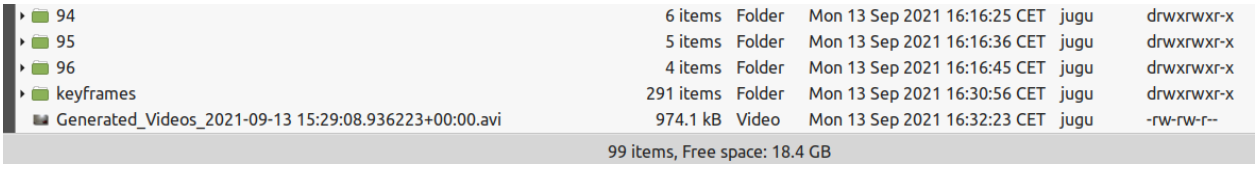

***Figure IV.28:** The final generated files after the MVS process*

In Figure IV.29 indicates different screenshots taken to see different generated number images and videos and their sizes accordingly.

- Number of received frames: **1076** with a size of ***484.2 MB:*** *(Screenshot **A**);*
- Number of keyframes after MVS process: **295** with a size of ***139.7 MB:*** *(Screenshot **B**);*
- Final generated video: 9 seconds video with ***974.1KB:*** *(Screenshot **C**);*

As we can see, the result is really impressive and extremely efficient in storage, we converted 484MB of data into 1MB approximately a ratio of 497% of storage compression.

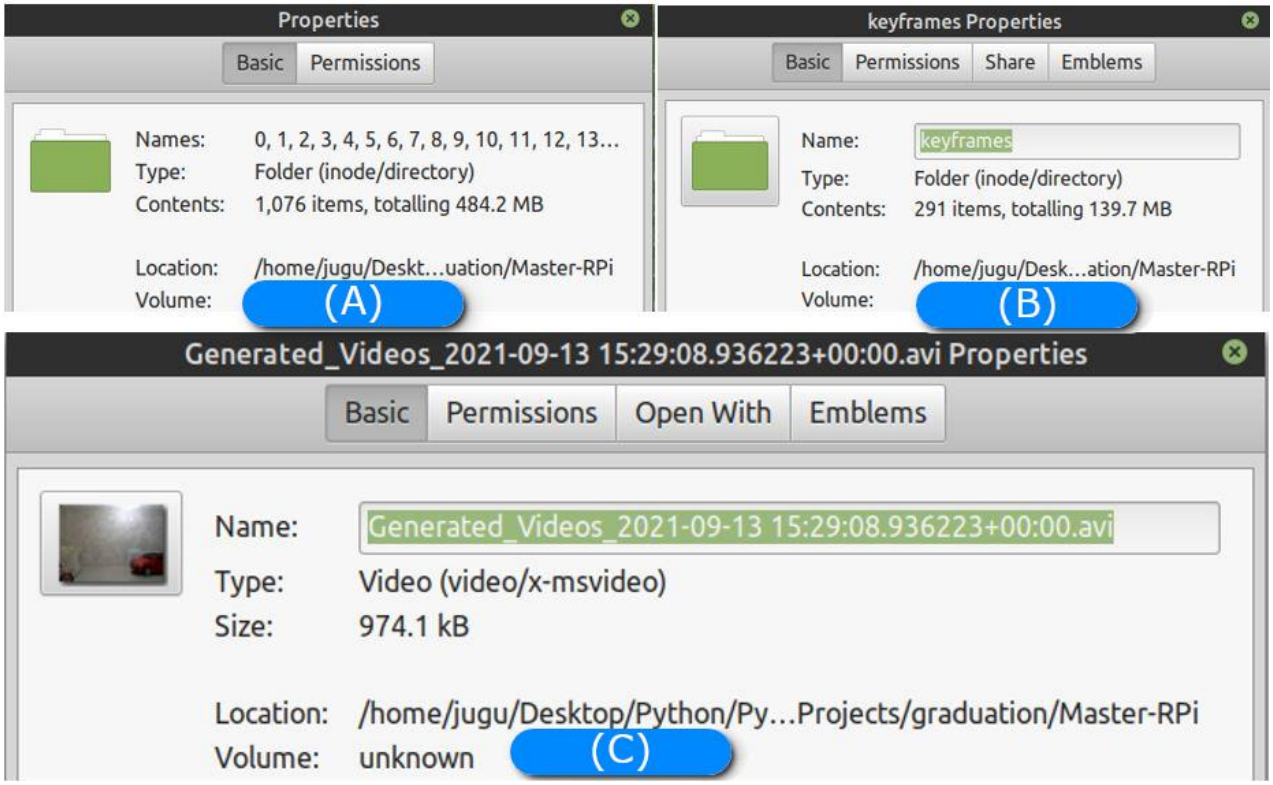

***Figure IV.29**: A set of screenshots indicating the number of generated images and their totaling sizes*

IV.4 Additional features

Beside to the regular features that we provided above, we added some extra features to make the project more interesting and useful. Some of the added features are:

IV.4.1 Remotely check the RPi cameras over HTTP:

Using this feature, we are capable to check the received frames in Real-Time via HTTP, this is possible by just using any web browser from any device (Smartphone, Tablet, PC…etc.).

We used FLASK framework to expose the frame over HTTP and display them in a web application built with HTML, CSS and JavaScript in frontend, in backend we used Flask, Flask

is a Python lightweight WISG web application framework used to serve web pages over HTTP, Figure IV.30 shows instantiation of Flask framework, to avoid using root access and extra configurations of firewall, we chose to use port 4747 instead of 80/443, this will also slightly add a layer of security.

```
362 ▶ if __name__ == '__main__':
363       # Instantiate a class
364       new_service = ReceiverService(open_port='tcp://0.0.0.0:5555') # frame receiver from Client RPi
365       print('[!] starting flask service...')
366   #   Preparing parameters for flask to be given in the thread
367   #   so that it doesn't collide with main thread
368       kwargs = {'host': '0.0.0.0', 'port': 4747, 'threaded': True, 'use_reloader': False, 'debug': False}
369   #   running flask thread
370       flask_thread = threading.Thread(target=app.run, daemon=True, kwargs=kwargs).start() # create a thread for evry HTTP client.
371       print("[+] Flask service has been started !")
372       time.sleep(1)
373       print('[!] starting Receiver service...')
374       new_service.start()
375       print("[+] Receiver service has been started !")
```

Figure IV.30*: Python code shows running Flask on separate thread on port 4747.*

After connecting to the web server using your Browser of choice, we will get the output as in Figure IV.31 where it displays the initiation or Flask service then starting the receiver deamon witch is continuously waiting for new Client Raspberry-Pi connection and receive frames.

```
/home/jugu/Desktop/Python/graduation/bin/python /home/jugu/Desktop/Python/PycharmProjects/graduation/Intelligent-Embedded-Vis
[!] starting flask service...
[+] Flask service has been started ! * Serving Flask app 'receiver_service_jugu' (lazy loading)

 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.55.176:4747/ (Press CTRL+C to quit)
[!] starting Receiver service...
starting the service...
Waiting for Clients Raspberry-PI connections...
Frame received from :anonymous - stored in :  ../Master-RPi/0/frame_145_v1.png
Creating directory....
```

Figure IV.31*: Python output logs shows the execution of the services.*

Figure IV.32 shows the render of the web page after connecting via a web browser to IP address 192.168.55.175:80, We can also connect using the hostname of the Master RPi since mDNS is already install on Raspberry machine.

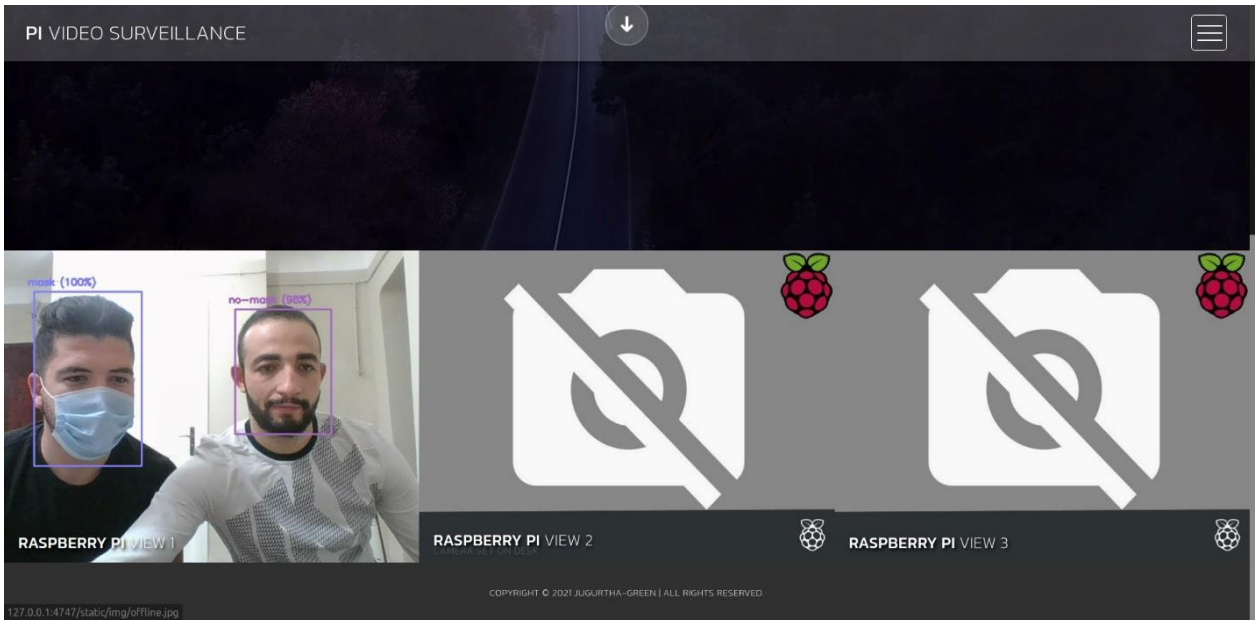

***Figure IV.32**: A screenshot shows the web page served by Flask through a web client.*

IV.4.2 Securing the Network via VPN:

In order to enhance the security of our IoT based network, Virtual Private Network (VPN) is considered as right solution for our case, VPN is an encrypted connection over the Internet from a device to a network. The encrypted connection helps ensure that sensitive data is safely transmitted. It prevents unauthorized people from eavesdropping on the traffic and allows the user to access the whole Network from everywhere in the world. The smartphone will connect to the Raspberry Pi through the Internet and create a secured tunnel between the two devices, so that we can access any service provided by the IoT based network (Like accessing client-RPi cameras via Web).

The VPN will secure all transmissions between RPis (client-RPi and master-RPi) even transmissions between User device (Laptop, Smartphone, tablet…etc) and the master RPi via HTTP or SSH. Figure IV.33 shows how VPN works.

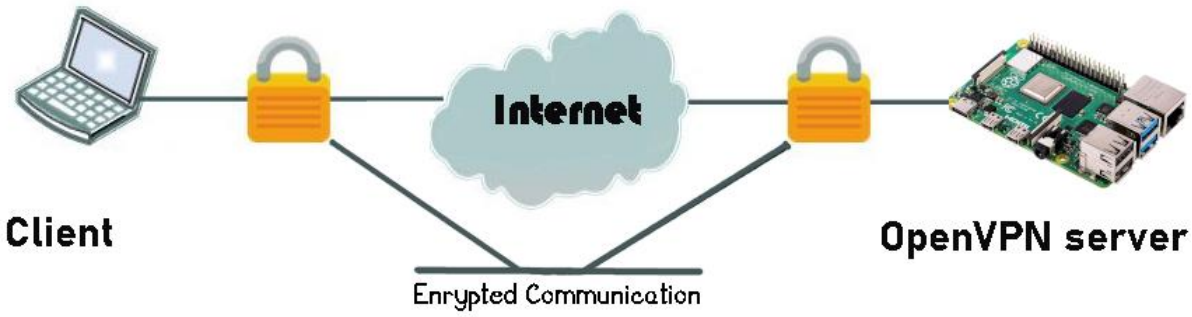

***Figure IV.33**: VPN Process*

To setup our VPN system, we need to install OpenVPN Server package on the Master RPi, OpenVPN is a virtual private network (VPN) system that implements techniques to create secure point-to-point or site-to-site connections in routed or bridged configurations and remote access facilities. It implements both client and server applications.

It uses the OpenSSL encryption library extensively, as well as the TLS protocol, and contains many security and control features. It uses a custom security protocol that utilizes SSL/TLS for key exchange. It is capable of traversing network address translators (NATs) and firewalls.

We also need Free Dynamic DNS (DDNS) hosting since our internet service provider doesn't provide us a static IP address. ***freedns.afraid.org*** provides us a free hostname (our case "***jugu-rpi4.mooo.com***" like shown in Figure IV.34) that redirects traffic to our IP address, even after a change. This way, we can configure our VPN client with **jugu-rpi4.mooo.com** instead of our dynamic IP address.

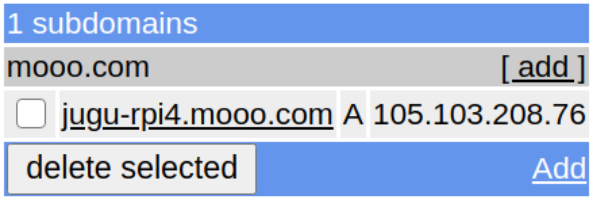

***Figure IV.34**: Subdomain used to access RaspberryPI over internet*

During the installation of OpenVPN on the Raspberry-PI, we configured the service to use Port **1194** on protocol **UDP** since we need speed while transmitting the keyframes. Figure IV.35 shows one of steps in the installation process.

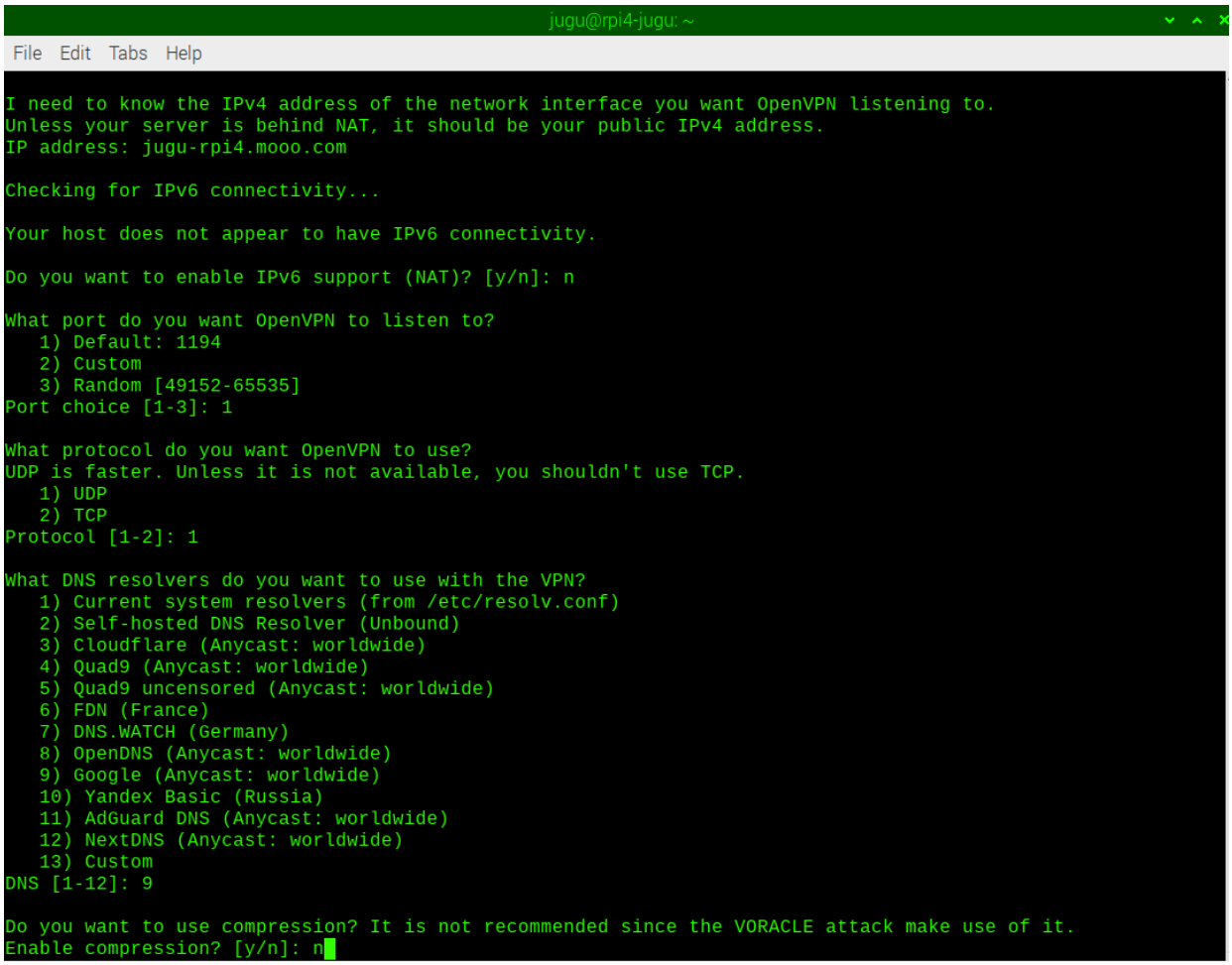

Figure IV.35*: VPN Configuration*

After the installation is completed, there is an extra step left to do, our Raspberry Pi is not directly accessible via Internet, but it's located behind a router. So we need to configure this router to redirect the VPN connections to our Raspberry PIs using port forwarding, this configuration depends from a router to another.

Finally, we need to configure our Raspberry-PI's firewall to open port 1194-UDP to allow incoming connections, this configuration is shown in Figure IV.36:

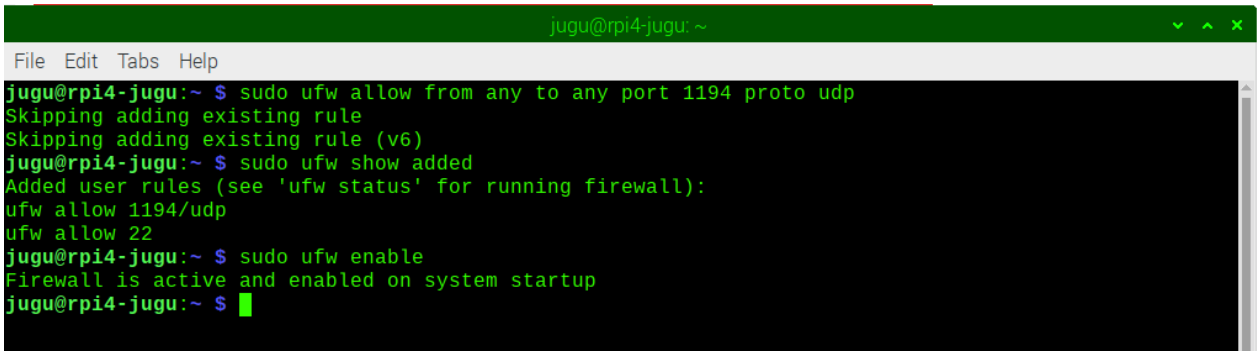

Figure IV.36*: Raspberry pi firewall configuration*

At the end, we will have a fully functional VPN server where connect to it using any device like smartphone, tablet, PC…etc. (Figure IV.37):

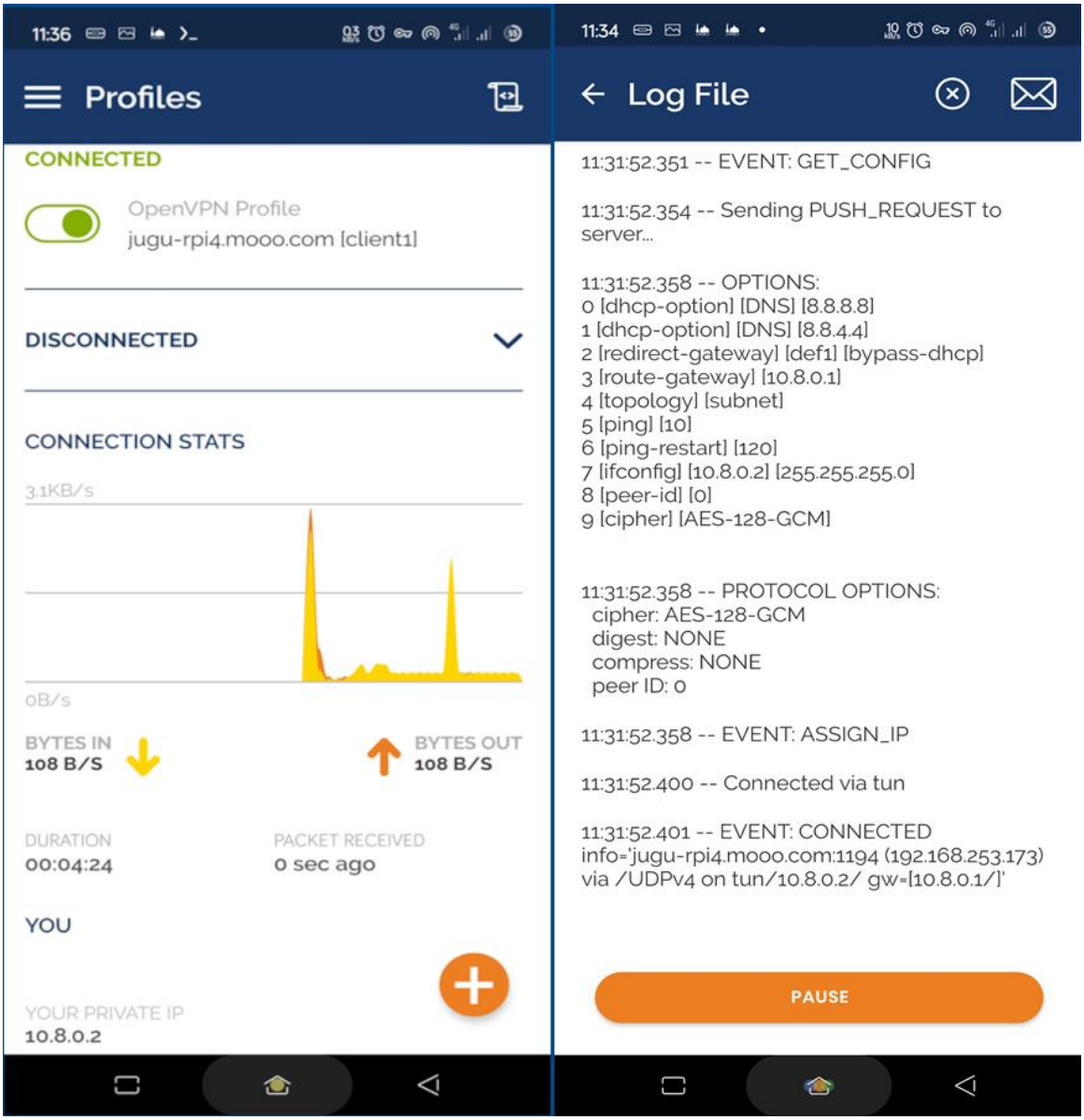

***Figure IV.37**: screenshot of smartphone connected to the master RPI's VPN server*

IV.5 Conclusion

In this last chapter, we described our system then explained its process steps, after that we had a look to the programmed application and briefly explained some of its functionalities. Finally, we closed this chapter with some additional features that can enhance the system usability.

Conclusion

In the modern age, surveillance networks are installed almost everywhere. These networks generate daily videos 24 hours a day with significant redundancy, this which wastes storage resources and makes them difficult to analyze. Motivated through these challenges, we proposed a tool for summarizing multi-video effective views based on light weight CNN.

Our job was to capture multi-view video (MVV) data. Each client Raspberry Pi (RPi) detects target in frames via light-weight CNN model, analyses these targets, and searches for suspicious objects (it can be any specific target) to generate alert in the IoT network (like police station). The frames of each client RPi are encoded and transmitted with to master RPi for final MVS. The proposed project can be used in industrial environments for various applications such as security and smart transportation even in smart-homes and can be proved beneficial for saving resources

Outlook: Although good results have been achieved, the work can be improved:

- We have used YOLO v4 tiny with darknet framework, this can be latter updated with YOLO v5 tiny with PyTorch.
- Add the possibility to switch between different model in real time without having to stop the script completely and edit it.
- Add more features and control through web application rendered by Flask.
- Expand the python script with C/C++ to increase the overall performances.
- Add the possibility to rotate the camera through motors and an auto tracking option to track the concerned target.
- Improve the script performances by adding some optimizations.

Bibliography

[1] Antonio Carlos Cob-Parro, «Smart Video Surveillance System Based on Edge Computing,» *Sensors*, p. 20, 23 April 2021.

[2] Laboratory, CCTV Technology Handbook, Advanced Technology and Assessments Branch éd., North Charleston, SC 29419-9022, July 2013.

[3] A. Seldon, CCTV Handbook 2021, Technews Publishing (Pty) Ltd, 1st Floor, Stabilitas, 265 Kent Avenue éd., Box 385, Pinegowrie 2123, 2021.

[4] Khan Muhammad, "Intelligent Embedded Vision for Summarization of Multiview Videos in IIoT | IEEE Journals Magazine | IEEE Xplore," *Intelligent Embedded Vision for Summarization of Multiview Videos in IIoT*, pp. 2592 - 2602, April 2020.

[5] «Discover the Differences Between AI vs. Machine Learning vs. Deep Learning,» Available: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/ai-vs-machine-learning-vs-deep-learning#:~:text=Artificial%20Intelligence%20is%20the%20concept,algorithms%20to%20train%20a%20model.

[6] Mathworks, Introducing Deep Learning with MATLAB, Mathworks éd.

[7] J. Brownlee, «A Gentle Introduction to Computer Vision,» Machine Learning Mastery, Available: https://machinelearningmastery.com/what-is-computer-vision/.

[8] Joseph Redmon, «You Only Look Once: Unified, Real-Time Object Detection,» 2016.

[9] «Introduction to YOLO Algorithm for Object Detection,» Engineering Education (EngEd) Program | Section, Available: https://www.section.io/engineering-education/introduction-to-YOLO -algorithm-for-object-detection/.

[10] I. John Wiley & Sons, Exploring Raspberry Pi® Interfacing to the Real World with Embedded Linux®, 10475 Crosspoint Boulevard: Indianapolis, IN 46256, 2016-06-13.

[11] I. John Wiley & Sons, Learning Computer Architecture with Raspberry Pi®, 10475 Crosspoint Boulevard: Indianapolis, IN 46256, 2016.

[12] " Raspberry Pi Documentation," 6 2021. Available: https://www.raspberrypi.org/documentation/faqs.

[13] John Wiley & Sons, Raspberry Pi® Projects For Dummies®, 111 River Street, Hoboken, NJ 07030: Library of Congress Control Number: 2015942453, 2015.

[14] Raspberrypi.org, «Raspberry Pi Documentation - Raspberry Pi Hardware,» Raspberrypi.org, Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/dpi/.

[15] generator, «Introduction to OpenCV-Python Tutorials,» Available: https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html.

[16] jeffbass, «imageZMQ: Transporting OpenCV images,» 2018. Available: https://github.com/jeffbass/imagezmq#introduction.

[17] «What is NumPy? — NumPy v1.21 Manual,» Available: https://numpy.org/doc/stable/user/whatisnumpy.html.

[18] «Raspberry Pi Comparison: Which Pi is Right for My Application?,» Available: https://www.digikey.com/en/maker/blogs/2018/how-to-pick-the-right-raspberry-pi.

[19] «General overview of the Linux file system,» Tldp.org, Available: https://tldp.org/LDP/intro-linux/html/sect_03_01.html.

Appendices

Appendix (A) : Functions of Operating System

A high-level definition of an operating system is that it stands between a computer user and the computer hardware, enabling the user to use the computer's various resources without interfering with other users or with computer operation itself. Its major jobs can be broken down this way:

■ **Process management:** The OS launches individual threads of execution for its own needs and the needs of users. It allocates execution time on the CPU among executing threads. If the CPU has multiple cores, it distributes processes among the cores.

■ **Memory management:** The OS allocates memory to running processes, in most cases as separate memory spaces that are protected from interference by other processes. Through a technology called virtual memory, the OS allows the computer literally to use more memory than it actually has, by writing the least-used process memory out to disk when more memory is needed.

■ **File management:** The OS maintains one or more file systems, which allocate file storage space on disks and other mass-storage devices and manage the reading of data from files and the writing to and deletion of files.

■ **Peripheral management:** The OS manages access to system peripherals like keyboards, mice, printers, scanners, graphics coprocessors and (in cooperation with file systems) mass storage devices. This is generally done through specialized software interfaces called device drivers.

■ **Network management:** The OS manages the computer's access to external networks through a collection of standard methods called networking protocols. The protocols are implemented in one or more pieces of software that, taken together, are called the network stack.

■ **User account management:** All modern operating systems allow different users to have their own accounts on the computer. An account includes a unique login, a set of security rules called privileges and a private file space protected from manipulation by other users.

■ **Security:** Scattered throughout an OS are mechanisms to keep running processes from interfering with one another and with the OS itself. Much of OS security is done by defining rules that specify what processes and users can and cannot do. Certain users called administrators or super users have powers that ordinary users do not have.

■ **User interface management:** The OS manages user interaction with the computer through software mechanisms called shells. A shell may be as simple as a text command line in a terminal window, or it can be a full-blown windowed graphical environment like those used in Windows, Mac OS X and desktop implementations of Linux, including Raspbian OS on the Raspberry Pi [10].

Appendix (B) : Embedded Linux

Embedded Linux is used to convey the presence of an embedded system, a concept that can be loosely explained as some type of computing hardware with integrated software that was designed to be used for a specific application.

This concept is in contrast to the personal computer (PC), which is a general-purpose computing device designed to be used for many applications, such as web browsing, word processing, and game play. The line is blurring between embedded systems and general-purpose computing devices. For example, the Raspberry Pi (RPi) can be both, and many users will deploy it solely as a capable general-purpose computing device and/or media player. However, embedded systems have some distinctive characteristics:

- They tend to have specific and dedicated applications;
- They often have limited processing power, memory availability, and storage capabilities;
- They are generally part of a larger system that may be linked to external sensors or actuators;
- They often have a role for which reliability is critical (e.g., controls in cars, airplanes, and medical equipment);
- They often work in real time, where their outputs are directly related to present inputs (e.g., control systems).

Embedded systems are present everywhere in everyday life. Examples include vending machines, household appliances, phones/smartphones, manufacturing/ assembly lines, TVs, games consoles, cars (e.g., power steering and reversing sensors), network switches, routers, wireless access points, sound systems, medical monitoring equipment, printers, building access controls, parking meters, smart energy/water meters, watches, building tools, digital cameras, monitors, tablets, e-readers, anything robotic, smart card payment/access systems, and more [10].

Appendix (C) : Advantages and Disadvantages of Embedded Linux:

There are many embedded platform types, each with its own advantages and disadvantages. Here are some of the reasons why embedded Linux has seen such growth:

- Linux is an efficient and scalable operating system (OS), running on everything from low-cost consumer-oriented devices to expensive largescale servers.
- A huge number of open source programs and tools have already been developed that can be readily deployed in an embedded application. If you need a web server for your embedded application, you can install the same one that you might use on a Linux server.
- There is excellent open source support for many different peripherals and devices, from network adapters to displays.
- It is open source and does not require a fee for its use.
- The kernel and application code is running worldwide on so many devices that bugs are infrequent and are detected quickly.

One downside of embedded Linux is that it is not ideal for real-time applications due to the OS overhead. Therefore, for high-precision, fast-response applications, such as analog signal processing, embedded Linux may not be the perfect solution. However, even in real-time applications, it is often used as the “central intelligence” and control interface for a networked array of dedicated real-time sensors [18].

In addition, there are constant developments underway in real-time operating systems (**RTOS**) Linux that aim to use Linux in a preemptive way, interrupting the OS whenever required to maintain a real-time process [10].

Appendix (D) : Booting the Raspberry Pi:

The first thing we should see when you boot a desktop computer is the *Unified Extensible Firmware Interface (UEFI)*, which provides legacy support for BIOS (Basic Input/Output System) services. The boot screen displays system information and invites you to press a key to alter these settings. UEFI tests the hardware components, such as the memory, and then loads the OS, typically from the solid-state drive (SSD)/hard drive. Therefore, when a desktop computer is powered on, the UEFI/BIOS performs the following steps:

1. Takes control of the computer's processor;
2. Initializes and tests the hardware components;
3. Loads the OS off the SSD/hard drive.

The UEFI/BIOS provides an abstraction layer for the OS to interact with the display and other input/output peripherals, such as the mouse/keyboard and storage devices. Its settings are stored in NAND flash and battery-backed memory.

Appendix (E) : The Raspberry Pi Bootloaders:

Like most embedded Linux devices, the RPi does not have a BIOS or battery backed memory by default.

Instead, it uses a combination of bootloaders. Bootloaders are typically small programs that perform the critical function of linking the specific hardware of your board to the Linux OS:

- They initialize the controllers (memory, graphics, I/O);
- They prepare and allocate the system memory for the OS;
- They locate the OS and provide the facility for loading it;
- They load the OS and pass control to it.

The bootloader for embedded Linux is a custom program that is tailored for each and every board type, including the RPi. There are open source Linux bootloaders available, such as Das U-Boot ("The" Universal Bootloader), Grub2, LILO…etc.; that can be custom built, given detailed knowledge of the hardware description of the embedded Linux platform.

The RPi uses a different approach: It uses efficient but closed-source bootloaders that were developed specifically for the RPi by Broadcom. These bootloader and configuration files are located in the */boot* directory of the RPi image:

*ls -ltr *.bin start.elf *.txt *.img fixup.dat*

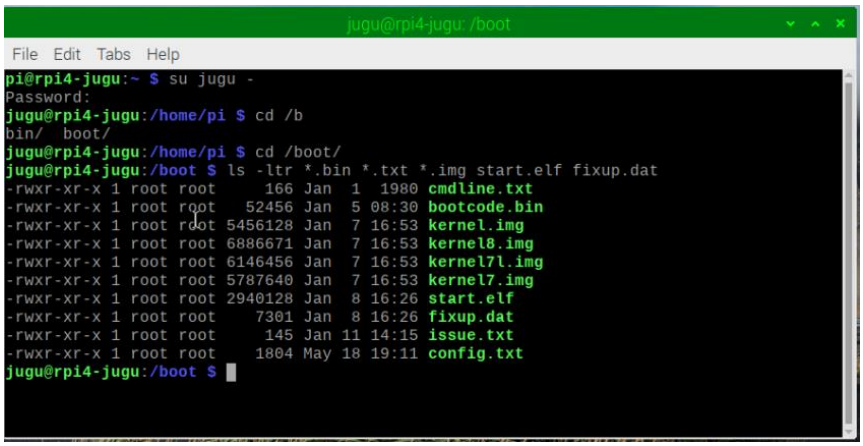

Figure E.1*: The output command to show the configuration files located in the /boot directory*

Appendix (F) : The boot sequence of the Raspberry Pi:

1. **Stage 1:** boot is in the on-chip ROM. Loads **Stage 2** in the L2 cache;
2. **Stage 2:** is *bootcode.bin*. Enables SDRAM and loads **Stage 3**;
3. **Stage 3:** is loader.bin. It knows about the " *.elf* " format and loads *start.elf***;**
4. *start.elf* loads *kernel.img*. It then also *config.txt***,** *cmdline.txt* and *bcm2835***;**
5. *kernel.img* is then run on the ARM.

Everything is run on the GPU until kernel.img is loaded on the ARM.

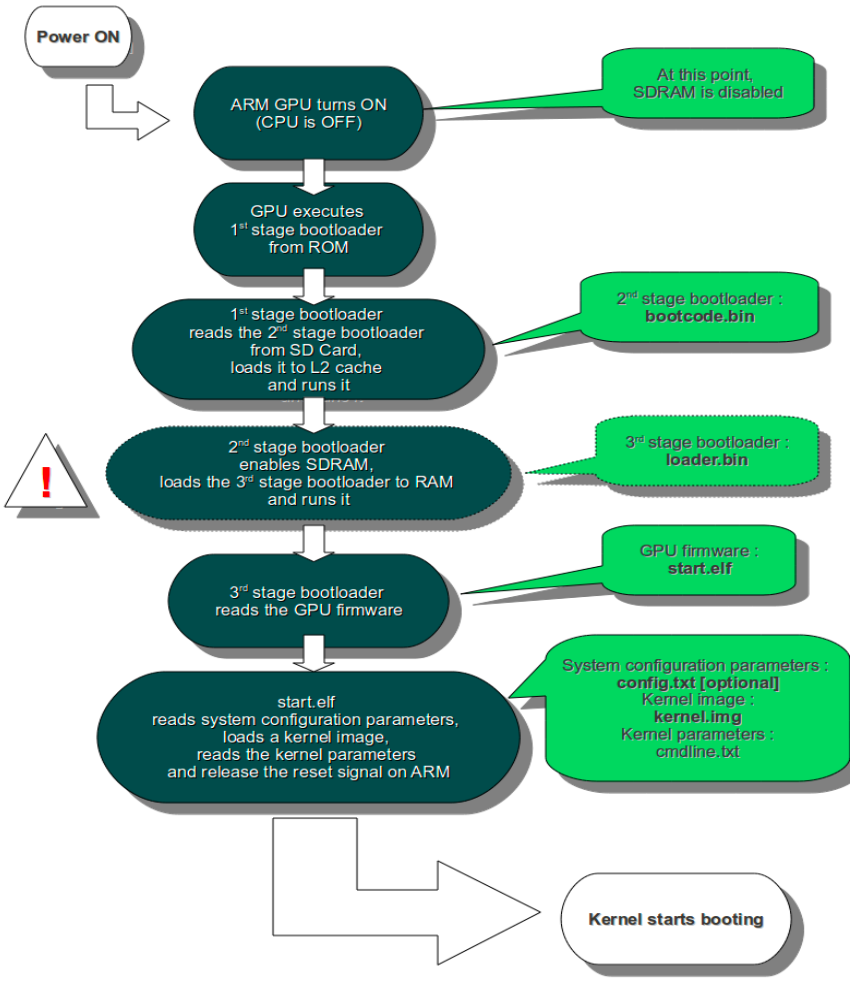

***Figure F.1**: The boot sequence of the Raspberry Pi in general*

Appendix (G) : <u>Multiple Cores:</u>

Modern CPUs often have more than a single execution core. A core is a separate and almost entirely independent engine that executes machine instructions. At the time of writing, CPUs with two, four and eight cores are common in the personal computing world. Each core executes processes independently, but all cores share system resources like memory. The operating system controls the use of all cores in a system, just as it controls everything else. The OS typically runs in one core, and parcels processes out to the other core(s) as needed.

Appendix (H) : The Super User:

On Linux systems, the system administrator account has the highest level of security access to all commands and files. Typically, this account is referred to as the ***root*** account or *superuser*. Under Raspbian/Debian, this user account has the user name ***root***, but it is typically disabled by default; however, we can enable it by typing ***sudo passwd root*** from a shell that is logged in with the **pi** user account (*The naming of the user account as "**root**" is related to the fact that it is the only user account with permission to alter the **top-level root directory "/".***).

Appendix (I) : The Linux Root Directory

Exploring the Linux file system can be mandatory for new Linux users. If we go to the top-level directory using **cd /** on the RPi and type **ls**, you will get the top-level directory structure, of the following form:

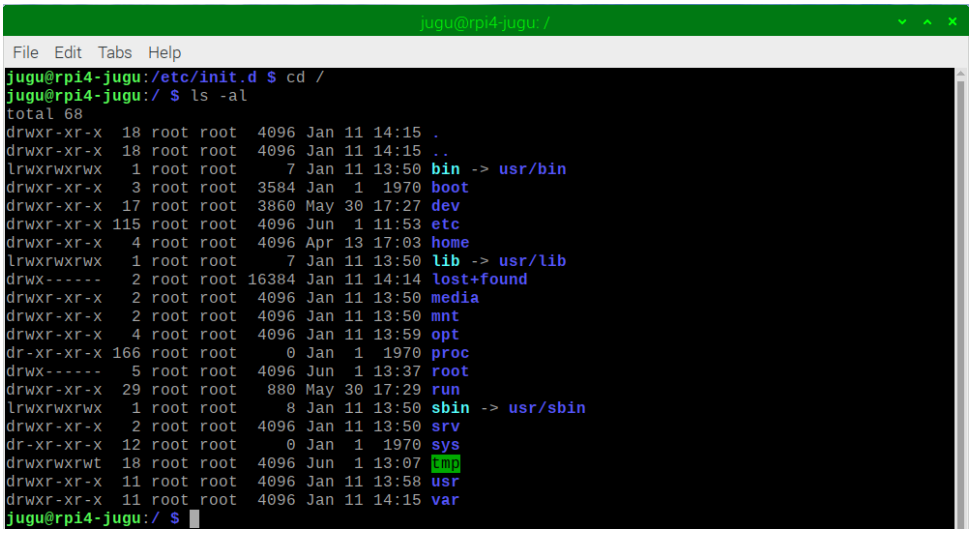

Figure I.1: *Linux command shows root directory.*

Each of these directories has a role, and if you understand the roles, you can start to get an idea of where to search for configuration files or the binary files that you need [19].

Directory	Content
/bin	Common programs, shared by the system, the system administrator and the users.
/boot	Contains the files for booting the RPi.
/dev	Contains the device nodes (linked to device drivers).

Directory	Content
/etc	Configuration files for the local system.
/home	Contains the user's home directories (***/home/pi*** is the **pi** user home).
/lib	Library files, includes files for all kinds of programs needed by the system and the users.
/lost+found	Every partition has a lost+found in its upper directory. Files that were saved during failures are here.
/mnt	Used typically for mounting temporary file systems.
/media	Used for mounting removable media, such as micro-SD cards.
/net	Standard mount point for entire remote file systems
/opt	Typically contains extra and third party software.
/proc	Virtual filesystem providing process and kernel information as files.
/root	The home directory of root account under the Raspbian and Debian Linux distributions.
/sbin	Contains executables for root user (superuser) system management.
/srv	Stores data related to ftp, web servers, rsync, etc.
/tmp	Temporary space for use by the system, cleaned upon reboot.
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files and temporary files created by users, such as log files.
/sys	Contains a virtual file system that describes the system.
/run	Provides information about the running system since the last boot.

Table I.1*: Briefly describes the content of each top-level Linux subdirectory*

Appendix (J) : OIDv4_Toolkit setup and usability

In the training process, we first installed a command line program **OIDv4_Toolkit** that will let us easily download specific classes from the Google **Open Images Dataset.**

Figure J.1, shows how OIDv4 download the given class names:

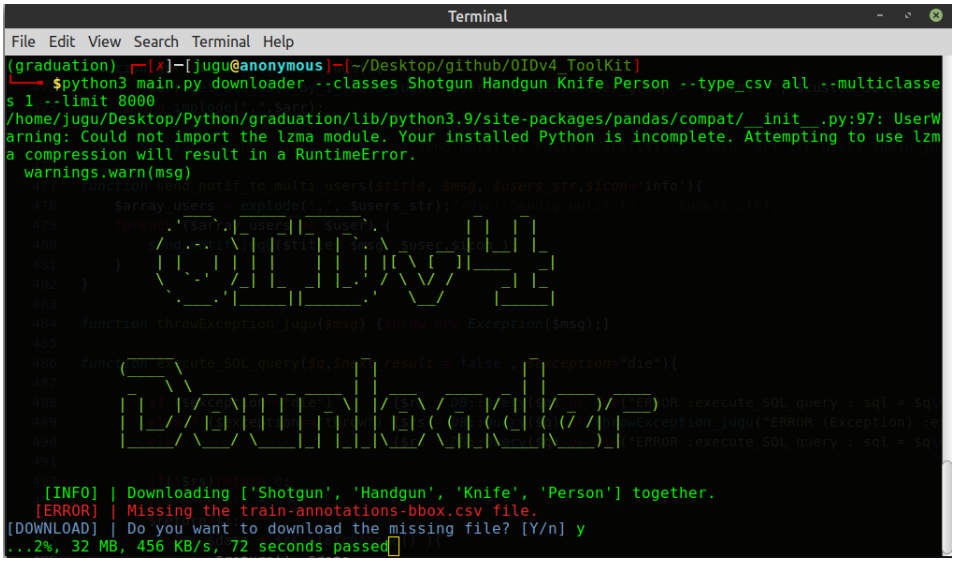

Figure J.1*: Downloading targeted classes from Open-Image-Dataset using OIDv4_Toolkit*

Appendix (K) : Application structure:

This section present and explain the directory structure for the project, Figure 52 shows the output of command "*tree*":

- **The folder "models":** contains our trained models, each subfolder has class names (with extension *".**names***"), the configuration file (with extension "**.cfg**") and the trained model as a binary file.
- **The folder "templates":** contains the HTML view file that will be rendered by Flask framework, this will allow us to remotely monitor the Client RPi camera via HTTP.
- **CoDec_jugu.py:** Python class responsible of compressing and decompressing the frames;
- **Complexity_jugu.py:** Python class witch calculate the complexity of a single frame;
- **Entropy_jugu.py:** Python class witch calculate the Entropy of a single frame;
- **LICENSE:** The license file which is Copy-Righted in our case;
- **main_Client_RPi_jugu.py:** Python script executed on Client RPi and its function is to apply the Object-Detection process, annotate the frames and send them to Master RPi;
- **main_Master_RPi_jugu.py:** Python script that applies MVS on the received salient frames;
- **motion_detector_jugu.py:** Optional Python class used to add motion detection to main_Client_RPi_jugu.py;

- **object_detector_jugu.py:** Python class responsible of object detection using a specified YOLO model;
- **README.md:** The readme file which explains different steps to install the system;
- **receiver_service_jugu.py:** A daemon Python script used to receive frames from Client RPi and store them as PNG images.

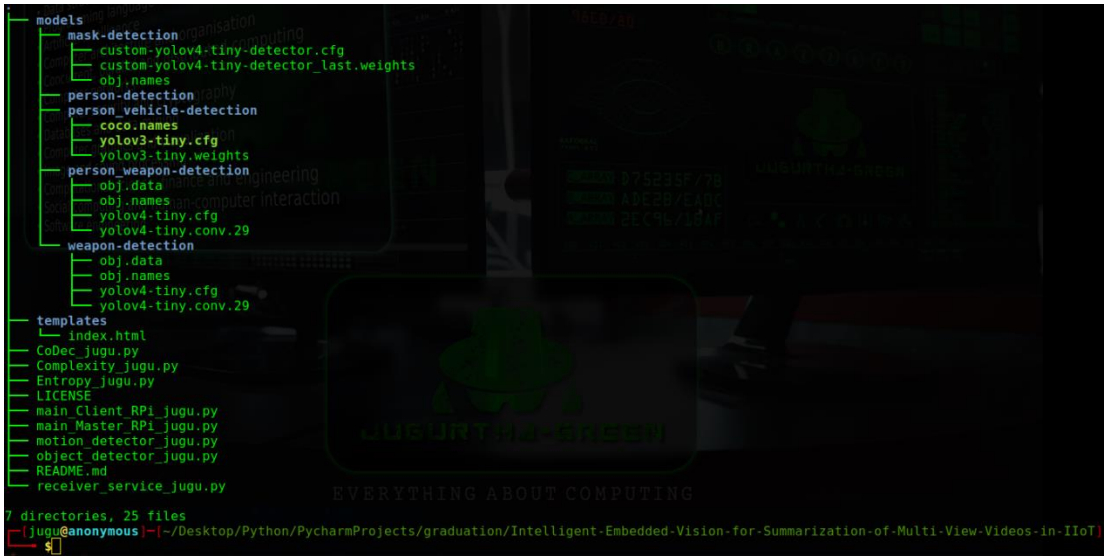

Figure K.1: *Screenshot shows the directory structure for the project*