POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Telecommunications Engineering

Camera Identification and Activity Classification from encrypted traffic: a Machine Learning approach for IoT Forensics

Relatore: Prof. Alessandro Enrico Cesare Redondi

Tesi di laurea di:
Luca Ferraro
Matr. 939443

Anno Accademico 2020 - 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital forensics is becoming more and more challenging due to the tremendous increase in computing devices and computer-enabled paradigm, providing new challenges to the distributed processing of digital data.
The increasing utilization of cloud services in their day-to-day operations by organizations, and the heightened emergence of smart device utilization means that digital forensic investigations involving such systems would involve more complex digital evidence acquisition and analysis.
Internet of Things (IoT) is the most evident field in which this challenge has become and still is becoming always more evident. The challenge is to precisely analyse large volumes of data in timely manner and collect forensic evidences related to crimes being investigated, while detecting the presence of IoT activity.

1.1 IoT background

The Internet of Things (IoT) is a concept coined to cover the interconnected infrastructure and utilities that are increasingly occurring. IoT is the interconnection of uniquely identifiable embedded computing devices within the existing Internet infrastructure. Such computing devices can be placed in any device, from smart meters and devices like video cameras, TVs or speakers, to remote sensors for gas and oil utilities. The key issue is the fundamental problem with the interconnection of all of these devices of the IoT world.
The IoT doesn't replace the ICT or operational technology networks; rather, it enhances these networks and relies on them in many ways. Recognizing all these aspects working together, cyber security and physical security solutions must also work together with a coordinated focus on threats. Also, considering the current amount of IoT devices and that this number is further increasing, particular attention must be paid to transportation, storage, access, and processing of the huge amount of data generated by these devices. Processing large quantities of IoT data will proportionately increase workloads of data centres, leaving providers facing new security, capacity and analytics challenges. Handling this data conveniently is a critical challenge, as the overall application performance is highly dependent on the properties of the data management service.
The IoT represents the seamless merging of the real and digital world, with new devices being created that store and pass around data. As a forensic analyst this has many consequences. On the one hand, one must find new ways to retrieve and secure this data making sure that there has been no tampering with the evidence. On the other hand, sometimes it could be useful to be able to collect and use the data for incriminate someone or providing an alibi for a crime that they may have

or may have not committed.

1.2 Digital forensics and IoT forensics

The discipline of digital forensics is a branch of the traditional forensic science concerning the uncovering and interpretation of electronic data. Digital forensics professionals deal with the identification, collection, recovery, analysis and preservation of digital evidences found on various type of electronic devices.
The IoT forensics could be perceived as a subdivision of the digital forensics; anyway, while the digital forensics discipline has long been in both academia and industry, the world of IoT forensics is a relatively new and unexplored area. The purpose of IoT forensics is similar to the one of digital forensics, so to identify and extract digital information in a legal and forensically sound manner. Besides from a particular IoT device, forensics data could be gathered from the internal network (e.g., a router) or from the cloud. What follows is a subdivision of the IoT forensics into three categories: IoT device level, network forensics and cloud forensics (Figure 1.1).

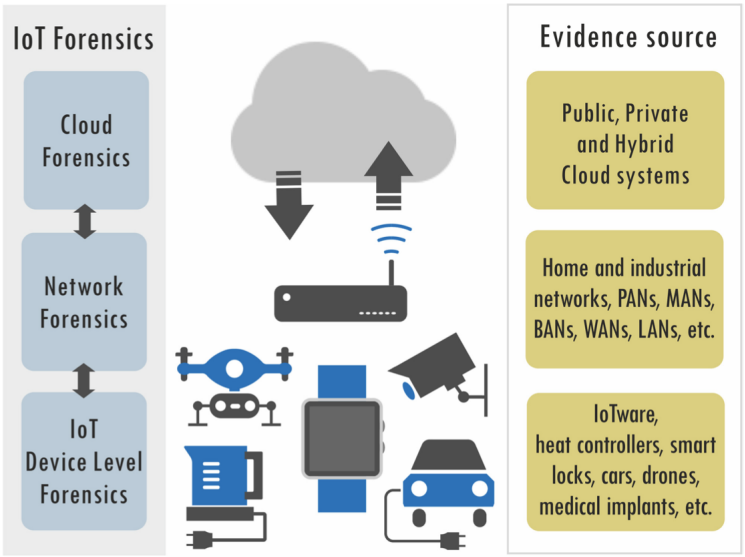

Figure 1.1: IoT forensics levels

As it can be seen in Figure 1.1, the fundamental difference between digital and IoT forensics can be seen in terms of evidence sources: while in digital forensics the usual objects of examination are computers, smartphones, tablets, servers or gateways, in IoT forensics the source of evidence is much more wide-ranging, including infant or patient monitoring systems, smart speakers, smart cameras, in-vehicle infotainment systems, traffic lights and even medical implants in humans or animals.

1.3 The importance of IoT forensics

There are may reasons why IoT forensics is needed.

- **Extensive attack surface** Just like machine-to-machine technology, IoT devices are particularly vulnerable to cyber-attacks types of communications. This is associated with the large number of terminals and embedded hardware

which increases its attack surface and puts it at higher risk.
IoT devices with public interface are exposed to a greater risk level, since they can bring a malware to the private network from a less secure public space. Commonly seen incidents includes identify theft and data leakage, accessing and using the internet connected printers, node tampering, commandeering of cloud-based CCTV units, SQL injections, phishing, insurance related frauds, cyberbullying, ransomware and malware targeting specific appliances such as VoIP devices and smart vehicles.
Moreover, cyber attacks could also have a large-scale nature and affect global enterprises or create chaos in stock marke.

- **New cyber-physical security threats** By using the power of IoT technology, virtual crimes can step across the limit of cyberspace and threaten humans life. For example, a smart lock can be programmed to unlock if a particular device is detected by the wireless network of a building, which could allow a criminal to access someone's home or office. Another example of possibly lethal scenario is the one in which a smart lock is reprogrammed to lock when a fire or a gas leak is detected. And these situations could be both the result of an intentional attack or consequence of a malfunction of the system.

- **Digital traces** A digital trace is a piece of information (stored on smartphones or IoT devices) which is able to prove or disprove certain hypothesis, therefore helping the forensic professionals to find answers and reconstruct a crime scene.
For example, digital traces may give information about when a smart home alarm was disabled and a certain door was opened, or by looking at the information gathered from a smoke detector it could be possible to determine the exact moment a fire took place, or again wearable devices lime smartwatches or fitness trackers can be used to identify a person via their biometric information.

1.4 Where IoT forensics is needed the most

The IoT market has and will continue to experience an exponential growth over the years.
As shown in Figure 1.2, the IoT market is grown from 157.05 billions USD in 2016 to 337.05 billions USD in 2019 and will grow up to 711.72 billions USD in 2026 (prediction according to a market study report made on 2019).

Cloud service providers have seen this as an opportunity to establish new business models; as a consequence, the forensics-as-a-service (FaaS) paradigm has started. However, securing the whole IoT infrastructure is not an easy task. Indeed, IoT communications consist of endless number of protocols, device capabilities and standards; therefore, IoT security relies upon securing each and every layer of the protocol stack in Figure 1.3.

Accordingly, IoT security and forensics tools are in high demand, and this applies to all the IoT-based domains like healthcare, smart home applications, industrial machines, supply chain and inventory management, smart grid, surveillance and smart cities.
The most appealing IoT domain for attackers are for sure industries, since they rely on sensitive data for real-time decision making. Therefore, the most important fields in which IoT forensics expertize are needed are:

- **IoT forensics for smart cities and vehicle automation** Smart cities are cyber-physical ecosystems that optimize the usage of conventional city

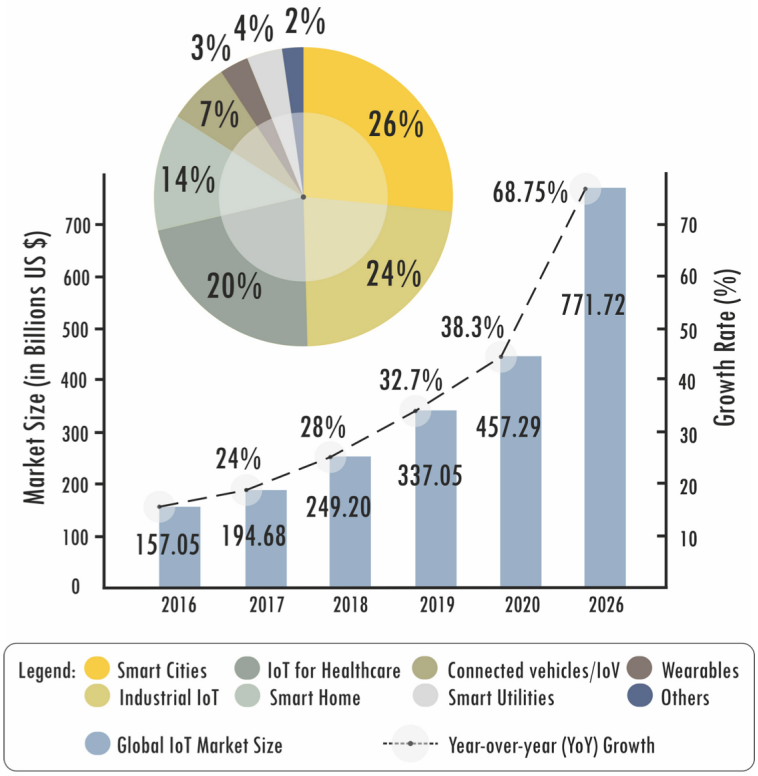

Figure 1.2: Global IoT market size (2016-2026)

infrastructure, therefore offering novel and convenient digital services to their citizens. One of the most important and effective attraction of smart cities is an intelligent transportation system, more precisely the so-called Autonomous Automated Vehicles (AAVs): equipped with sensors for Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications, smart cars and buses can provide to passengers, drivers or agencies the necessary information for efficient and safe city traffic.
In this scenario, especially interesting from a forensic point of view is the vehicle infotainment system, since it stores a vast amount of user related data (navigation history, call logs, contact list, ...): a forensics professional could use such information to identify liabilities in traffic accidents or cyber attacks. Moreover, in case of accident, they could examine the sensor records from the neighboring cars.

- **IoT forensics for smart home/office** Smart building applications support personalization by controlling over the living or working environment. A system formed by sensors and actuators aim at enhancing the comfort of the residents, even without their intervention.
  Common IoT devices for smart home/office are air conditioners, floor heating systems, refrigerators, washing machines, lights, smart locks, speakers, cloud cameras. All of these devices can be controlled over the internet for saving water or energy or other resources or just to check what is happening in the home/office.
  Here, particularly interesting for the forensics professional is understanding how such devices can be used for retrieving information on a user's life. For

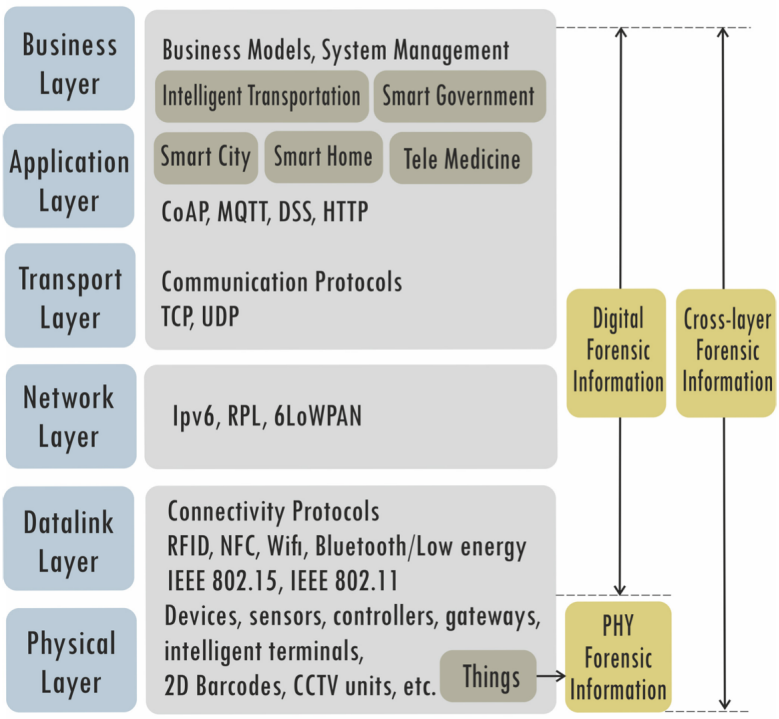

Figure 1.3: IoT layered architecture

example: is it possible to know who is using a smart speaker? Or is it possible to understand what is happening in the Field of View (FoV) of a camera? Answering to these questions could prove that someone specifically was inside a hose/office and that something particular was happening.

- **IoT forensics for healthcare** This is probably the most most vulnerable to major security attacks because of the cross-organization nature of IoT applications in this field as well as their heterogeneity, fragmentation and expanded attack surface.
  Consider for example a fitness tracker: a malicious actor could target one and use the gathered data for illicit financial gains by selling it, for instance, to insurance company or blackmailing the owner of the compromised device.
  Also, wearables like fitness bracelets have gained importance since they are designed for passively collecting data in background, thus providing a lot of information on a user's daily activity, and this information can be very important from a forensic point of view.

1.5 Aim of this thesis

The aim of this thesis project is exploiting a particular application of the IoT forensics world, the one related to the utilization of smart cloud cameras.
The aim of such devices is very simple and clear: they allow a user to check what's happening in the FoV of the camera just by using an app on their smartphone or tablet.
Of course, the proprietary of a camera can access to the full, complete and decrypted video stream transmitted by the camera since their smartphone/tablet is associated with the camera itself. But what if another user that may have or may have not

access to the decrypted video stream wants to check weather they can guess what's happening in the FoV of the camera using some machine learning (ML) algorithm? This is the question to which this work wants to answer.
This question is very interesting from a forensic point of view since a positive answer would imply the possibility to reconstruct with a certain accuracy the events that have happened in the room of a house, or in a office, or in whatever place in which an IP camera is placed for security reasons. Therefore, it's possible to use these data as evidence to support a certain thesis.
However, even if this looks very positive from a forensic point of view, it may sound worrying from a privacy point of view. Indeed, if it's possible for a forensic professional to retrieve such information, it may also be possible for an attacker to infer information that they can use for malicious purposes.

1.6 Structure of this work

In order to answer the central question that guides this work, this report will follow the following steps.
First of all, in Chapter 2, I will provide a brief description of some related works and the main differences between those works and this thesis project.
Then, in Chapter 3 I will describe the main reason why it can be thought that it's actually possible to say with a certain level of accuracy what is happening in front of a camera, that is the way video traffic is encoded.
Chapter 4 will then describe the cameras that have been used for the whole duration of the experiment and the system that has been built for collecting the data used to train and test different ML algorithms.
What will follow is, in Chapter 5, the description of the features that have been used as input for the ML algorithms and why such features have been chosen; also, I'll explained which activities I have tried to identify, how data have been actually collected and how the features have been extracted.
At this point, in Chapter 6 I will provide results about the accuracy of different ML algorithms.
Finally, in Chapters 7 I will respectively describe a possible approach used for a live analysis of the traffic sent by some camera with all the classification phases so to try to actually use what has theoretically been proved and the possible integrations of the this part of the work.
At the very end, in Chapter 8, I will derive some conclusions and briefly talk about some possible future works.

Chapter 2

Related works

In this chapter, I will briefly describe some of the works that have inspired this thesis project.

The starting point is a paper from 2015 redacted by Chris Wampler, Selcuk Uluagac, and Raheem Beyah whose title is *"Information Leakage in Encrypted IP Video Traffic"*. In this research, the authors show that information leakage is actually happening in video over IP traffic. In particular, they demonstrate that, by looking at some properties of the encrypted payloads of such a traffic, it's possible to detect motion and scene changes.
The experiments of this work are performed using different codes and cameras, therefore establishing a basis for detectability of events via packet timing.

On top of this work, other researches and experiments have been performed to check how this information leakage can be used to obtain some useful knowledge.

In particular, the work from 2016 by Hong Li, Yunhua He, Limin Sun, Xiuzhen Cheng and Jiguo Yu titled *"Side-Channel Information Leakage of Encrypted Video Stream in Video Surveillance Systems"* shows how to use the side-channel to retrieve knowledge on a user's basic activities of daily living just by looking at traffic size data of an encrypted video stream using the properties of difference coding techniques like MPEG-4 and H.264.
The results are validated using two commercial cameras for data collection. Data are captured and processed as time series data so to obtain a different capture file for each single time an activity has been performed. Finally, two machine learning approaches (K-NN classification and DBSCAN clustering) have been applied to check the level of accuracy with which activity identification can be performed.
This research tried to distinguish between for daily activities: dressing, hair-styling, moving and eating.

A more deep research were conducted in 2020 by Maria Stoyanova, Yannis Nikoloudakis, Spyridon Panagiotakis, Evangelos Pallis, and Evangelos K. Markakis; results are reported in a work with title *"User Behavior Classification in Encrypted Cloud Camera Traffic"*.
Here, the authors performed a work that is similar to the previous one in the sense that they both aim at reconstructing a user's daily routine, but here the focus is on distinguishing between more activities (dressing, reading, sweeping, opening/closing a door, drinking, sporting, watching TV, walking, combing hair) and on trying a lot of classification algorithms (Decision Tree, Gradient Boosting GT, K-NN, Logistic Regression, Perceptron, Random Forest, Multi-layer Perceptron, AlexNet) to ver-

ify how different machine learning classifiers behaves when it comes to distinguish among a lot of activities.
Here, results were validated using two commercial cameras (not the same of the previous work).

All of the works mentioned above have the purpose of using the properties of difference coding to retrieve information about a user's habits, but up to know the aim was academic, with the focus of the privacy issues that may be the consequences of the results of the works. But, is there a field in which this knowledge can be used for non malicious purposes?
The answer is yes: one of the possible fields in which having a way of understanding what have happened in the room of a house, or in an office, or in whatever place can be determinant is the forensic field. For the forensic professional, the possibility of obtaining some evidences from IoT devices such as IP cameras can be determinant in supporting a thesis.
This IoT forensics world is new and not so explored: there is a number of works and papers talking about its peculiarities with respect to traditional digital forensics, about the reasons why the development of the research in the IoT forensics world is important and about which are the main fields in which this new discipline is needed the most. An example is the a paper from 2020 redacted by Maria Stoyanova, Yannis Nikoloudakis, Spyridon Panagiotakis, Evangelos Pallis, and Evangelos K. Markakis having title *"A Survey on the Internet of Things (IoT) Forensics: Challenges, Approaches, and Open Issues"*.

All of the works mentioned above have have inspired this work in the sense that the results at the basis of this research and the results I have tried to obtained are strictly related to the ones of those previous jobs.
Anyway, there are many peculiarities in my jobs that makes it distinguishable from research already done:

- First of all, the perspective is different. In all the three papers I have reported, the eye is the one of an attacker who wants to infer information on a user's daily habits and behavior, so the focus is on the privacy implications of the results.
  Vice versa, this work uses a forensic approach: the aim is the one of understanding whether it's possible to obtain evidences that can be used by the forensic professional for supporting a certain thesis. Of course, the information we want to obtain are the same, but their scope and how they are used is completely different.

- Another fundamental difference is the approach used to analyze the collected data.
  In the previous works,data were first preprocessed to make activity segmentation, that is, a different capture file was created for each time interval in which an activity was detected (starting and ending instants of an activity are identified looking at the transmission rate of a camera); then, features were extracted on the whole capture corresponding to an activity and finally the classifiers were trained.
  In this research, there is no activity segmentation procedure so to be able to identify also when nothing is happening; therefore, features aren't extracted on the whole capture file but looking at small time windows of it. This way, we might be able to reconstruct the events that have took place in the FoV of a camera with more granularity (which depends on the length of the time window over which features are extracted).

- Finally, this work contains an identification phase that hasn't been carried out in any of the above mentioned papers: the identification of the camera that is transmitting. Indeed, those papers have performed activity identification separately for different cameras, but they haven't faced the problem of checking whether the same information used to distinguish between different activities can be used to distinguish between different cameras or not.

Chapter 3

Video encoding and information leakage

In this chapter, I will first describe how video streams are encoded and why and then how the properties of this encoding technique can be used to answer the question that guides this work.

3.1 The problem of the bandwidth

What we call a video is nothing more than a sequence of images in time. The number of images that are visualized in ones second is a metric called frame per second (fps): of course, the higher the number of fps of a video, the more fluid the vide is.
Each frame composing the video is made of pixels The number of pixels of an image defines how big the image is; indeed, the dimensions of an image is given in number of pixels.
Then, each pixel is simply a colored dot, so for each pixel of an image it must be defined how it's colored. A simple way of coding the color information of a pixel is through the RGB coding: each pixel is assigned with three numbers representing respectively the amount of red, green and blue in the pixel itself and the actual color is the linear sum of these three quantities. Usually, for each color it's used a resolution of 8 bits, that is, it's possible to distinguish between 256 levels of red, green and blue; therefore, for each pixel we need 24 bits to define its color.
Given all af this information, if we have a video in which each frame has height H and width W pixels and for which the frame rate is F, the requested bandwidth is $B = H * W * F * 24$ bits per second (bps). For the video resolution that nowadays are requested, this number is really huge. For example, a 4K video has $H = 2160$, $W = 3840$ and $F = 24$, which results in $B \approx 4.78 * 10^9 = 4.78$ Gbps. If we consider that the fastest rate we can achieve using a normal internet connection in our house in bounded at 1 Gbps, we immediately realize that transmitting always all the information of a video stream is impossible.

3.2 Difference coding

In oder to solve the problem of the bandwidth and allowing the transmission of video streams over the network without too many problems, a solution is encoding the video stream in a way that we don't always transmit a full image, but only the difference between two images. This kind of encoding is called difference coding.

There is a variety of difference coding algorithms and as long as the output format is obeyed each algorithm can be implemented in a variety of ways. What all of them have in common if that they organize a sequence of frame in a so called Group of Pictures (GoP). Within a GoP, a frame can be one of three types: an I frame, a P frame or a B frame. An I frame is a frame that is actually not encoded, which means it carries a whole picture, while P frames and B frames carries encoded information, in particular they are built by looking at the difference between the current P or B frame and other frame(s) (which one(s) can vary depending on the actual encoding algorithm) in the same GoP. Figure 3.1 shows a possible structure of a GoP with the relation between I, P and B frames.
By proceeding in this way, it's possible to save a lot of bandwidth since a full image (that is, an I frame) is transmitted only once in a while; the rest of the transmission consists in P and B frames, which are significantly smaller that I frames in terms of bytes.

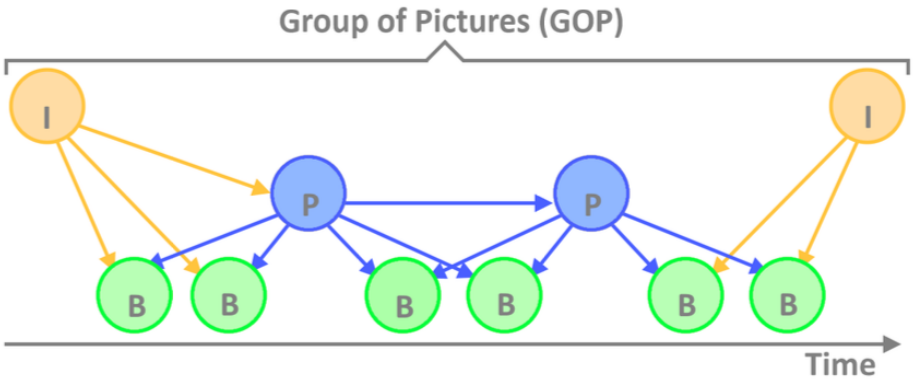

Figure 3.1: Possible structure of a Group of Pictures

3.3 Difference coding techniques and information leakages

At this point, it's clear that the strength of difference coding is that it allows to save a lot of bandwidth by transmitting only sometimes a whole image and by encoding the rest of the frames by looking at the different pixels between the current frame and another one.
This base property of the difference coding approach is also what I will use in this work for recognizing an activity that has took place in the FoV of a camera.
Let us now put ourselves in the perspective of an observer that has no access to the decrypted video stream of a smart cloud camera, that is we know there's such a camera that is transmitting a video stream (also, let us suppose we know its MAC address), but we can only observe its encrypted traffic (this can be done using a packet analyzer like *Wireshark*). The point is that, even if we cannot see explicitly the data contained in the packet (since they are encrypted), we still have access to some statistics about the video traffic such as the dimension of the packets and the inter-arrival time. Note that the level at which this information can be retrieved depends on if the observer is the proprietary of the network on which packets are analyzed or not: in the first case it's possible to access the information up to the application level, while in the second only up to the MAC level.
Now, if we remember how difference coding works, we can easily understand that

we can actually use the statistics we have to infer some information about what's happening in the FoV of a camera. Suppose for example that nothing is happening: since a camera is not moving, then, the differences among frames will be so small that the transmission will essentially consists in an I frame once in a while and then a series of very short P and/or B frames. Vice versa, if something is happening in the FoV of the camera, then there will be more pixels assuming different values over time, so the P and B frames will be larger.
This is the idea at the base of the whole work: different activities carried out in the FoV of the camera will produce encoded video streams with different statistics; therefore, we can use those statistics to retrieve some information and try to guess which activity is being performed in the FoV of the camera.

3.4 Information leakage: monitor mode and non monitor mode

As stated in the previous section, an observer who doesn't have access to the decrypted video stream but only to some outline information on it, can still retrieve some knowledge on the happenings the the FoV of a camera, and this knowledge can be obtained at two different levels: application or MAC.
There are two main reasons why it's interesting to check the actual knowledge that can be obtained by both the layers:

- First of all, at different layers an observer has access to different information, so it's of interest to verify if different information leads to the same conclusions about something happening the the FoV of a camera or to different ones.
- From a purely forensic point of view, evidences can in principle be found at different levels depending on the role of the observer with respect to the system. For example, if the observer is also the proprietary of the router through which the cameras send their traffic to the internet, then it can obtain information up to the application layer; vice versa, if the observer is completely external (that is, it has no access to the router to which the cameras are connected), it can only reach the information up to the MAC layer.
  Therefore, it's forensically really interesting to understand how evidences can be retrieved both at application and MAC layers.

3.5 Practical examples of information leakage

Here, I just put some images that support what has been stated in the previous section.
The images consist in graphs of the traffic size and of the inter-arrival time of the video stream of the three cameras that I have used in the whole work (one by Teckin, one by Tapo and one by Ezviz; more details and specifications in the next chapter).
In the graphs of the traffic size, I have considered:

- In non monitor mode, the length of the IP header plus the length of the header of the transport layer protocol and the actual length of the payload from the application level.
- In monitor mode, I have directly considered the length of the payload at MAC layer (since its the best we can do).

Each graph shows what happens for a single camera when there is an alternation of no activity and a some activity in the Fov of the camera (namely, someone moving

slowly, someone moving rapidly and a light event). Moreover, for each activity I report the graph resulting if data are collected both at application level and at MAC level; for consistency, data collection had happened simultaneously for the two versions and for all the cameras.
Also, the curves are smoothed using a moving average filter with no overlapping in which the dimension of the window was found by considering the average number of packets in each window of 0.75 seconds (this time window has been chosen empirically).
As we can see, the graphs are the proof of what has been stated before: different activities in the FoV of the cameras produce video streams with different shapes and therefore different statistics; as a consequence, it has to be possible to retrieve some kind of information.
In particular, there are some things that can be noticed:

- All the traffic size graphs present almost periodic peaks: the positions in time of these peaks correspond to the instants in which one or more I frames have been sent.

- In the time intervals between two peaks, the traffic size curves almost show two different behaviors: either a series of very short packets sent (that is, the curves are low in the y-axis), or longer packets are sent (so, the traffic size is high on the y-axis). These two different behaviors correspond respectively to the intervals in which there was nothing happening in the FoV of a camera or, vice versa, something was happening. Moreover, the actual dimension of the traffic size depends on the activity performed: the more eventful the happenings in the FoV, the more different pixels from a frame to another one, the more information to encode, the larger the traffic size. So, the graphs corresponding the the *Movement fast* activity will in general present a larger traffic size with respect to the ones of the *Movement slow* activity.

- If we also look at the graphs corresponding to the inter-arrival time trying to find a relation with the ones showing the traffic sizes, we can notice that in the intervals in which the traffic size increases, the inter-arrival time decreases. This means that, depending on the events in the FoV of a camera, the more the information to be encoded and sent, the higher the rate (larger packets and shorter inter-packet times) the cameras are sending this information to the internet. Therefore, looking at the rate could be a clever idea for understanding what's happening in the FoV of a camera.

- Another thing that can be noticed is that, if we fix an activity in the FoV of the cameras, the curves are not the same for different cameras: they have the same behavior (increasing traffic sizes and decreasing inter-arrival times when something happens), but not the exact same shape. This fact suggests that an activity identification classifier should be different for different cameras. Also, we may ask another question: since different cameras produce different curves for the traffic size and/or the inter arrival time, is it possible to use some statistics to identify also the camera that is transmitting? This is another question I'll answer in this work.

- All of the characteristics that I have just mentioned are much more evident in the graphs corresponding to the captures performed in non monitor mode than in monitor mode. This is something expected, since in non monitor mode the information we have on the nature of the packets are much more explicit (because we have access to all the layers up to the application layer). The implication of this is that, probably, the results we are going to obtain for the

two modes, monitor and non monitor, should be such that the ones in non monitor mode have to be at least as good as the ones in monitor mode.

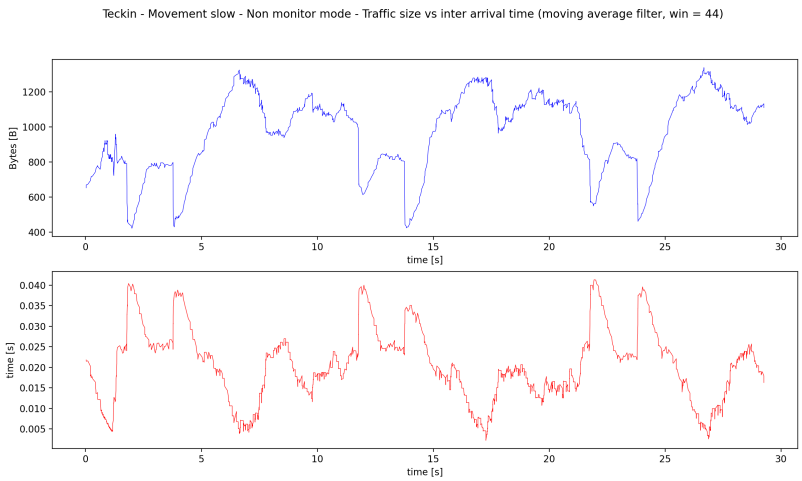

Figure 3.2: Teckin - movement slow - non monitor mode

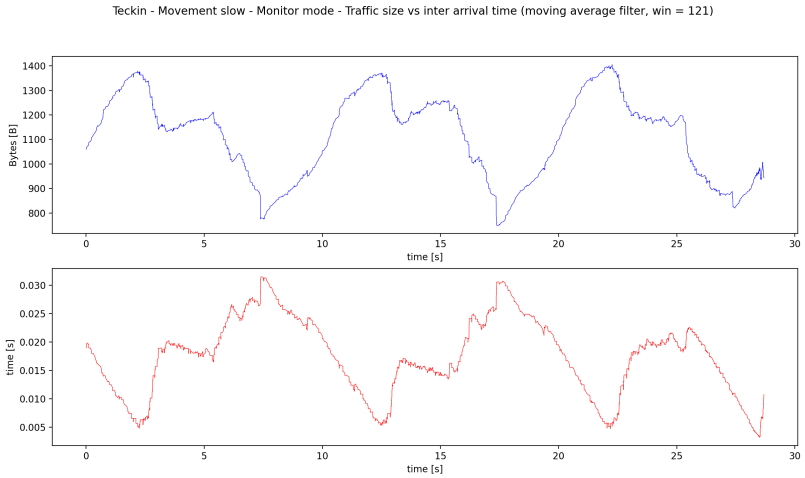

Figure 3.3: Teckin - movement slow - monitor mode

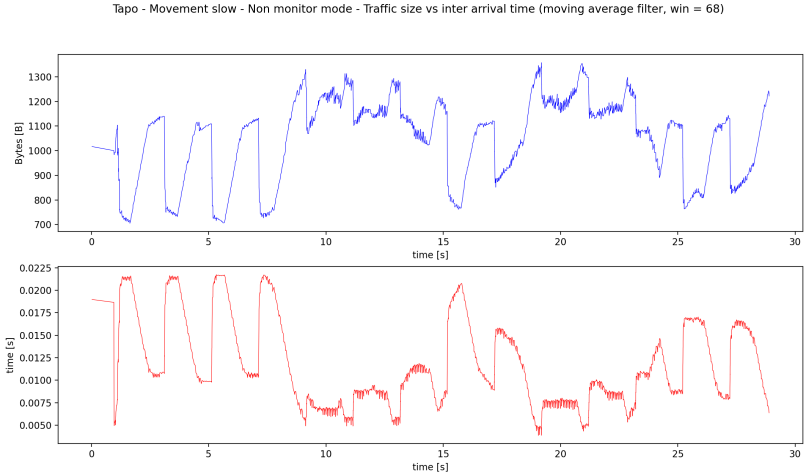

Figure 3.4: Tapo - movement slow - non monitor mode

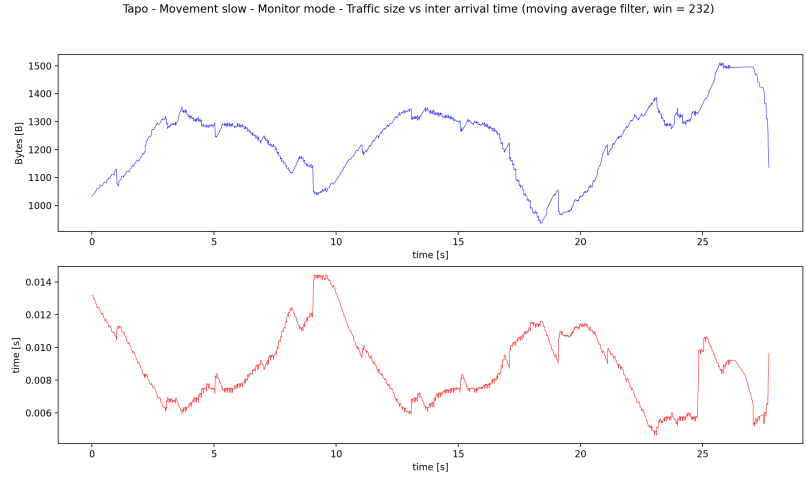

Figure 3.5: Tapo - movement slow - monitor mode

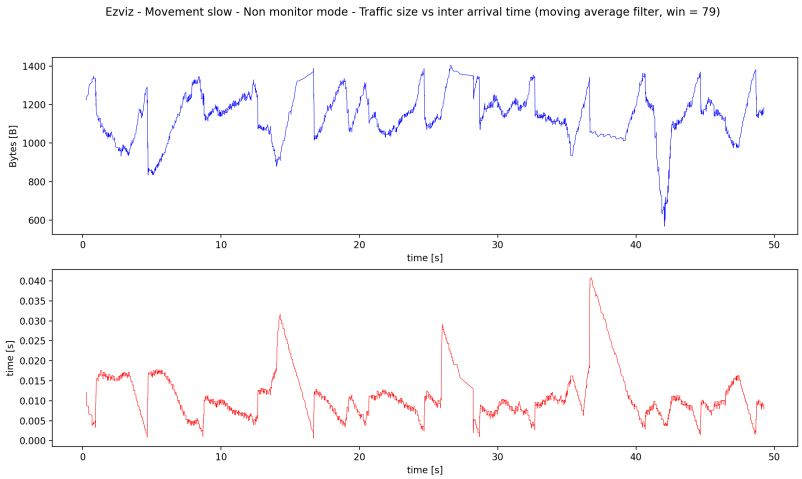

Figure 3.6: Ezviz - movement slow - non monitor mode

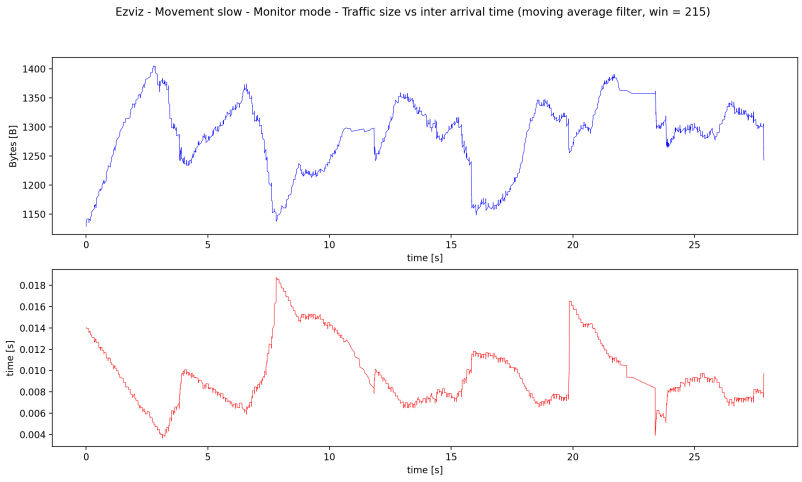

Figure 3.7: Ezviz - movement slow - monitor mode

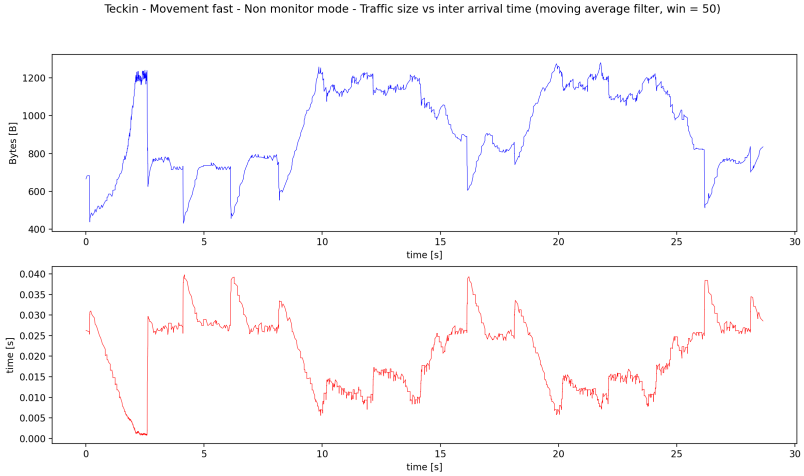

Figure 3.8: Teckin - movement fast - non monitor mode

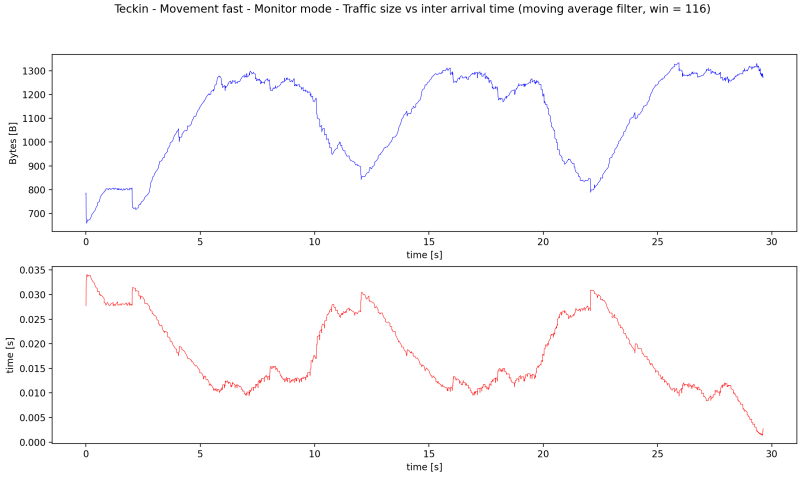

Figure 3.9: Teckin - movement fast - monitor mode

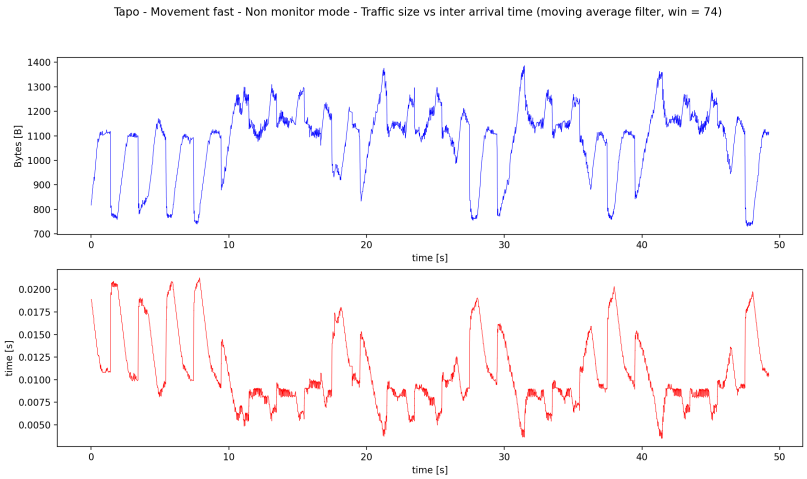

Figure 3.10: Tapo - movement fast - non monitor mode

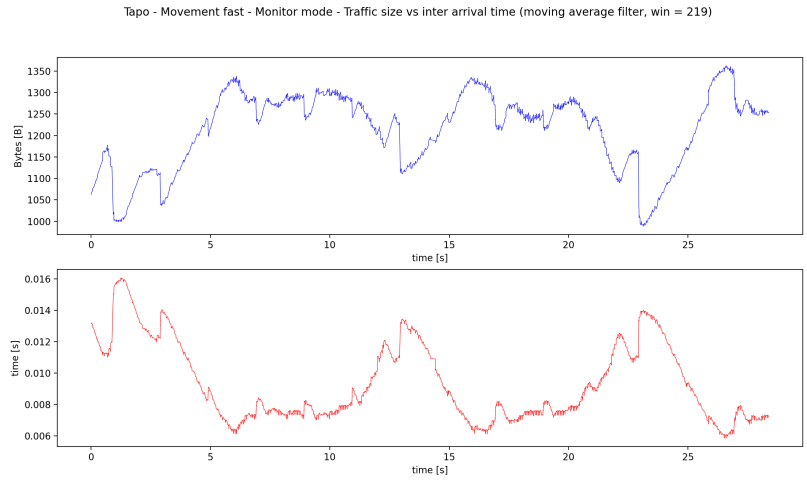

Figure 3.11: Tapo - movement fast - monitor mode

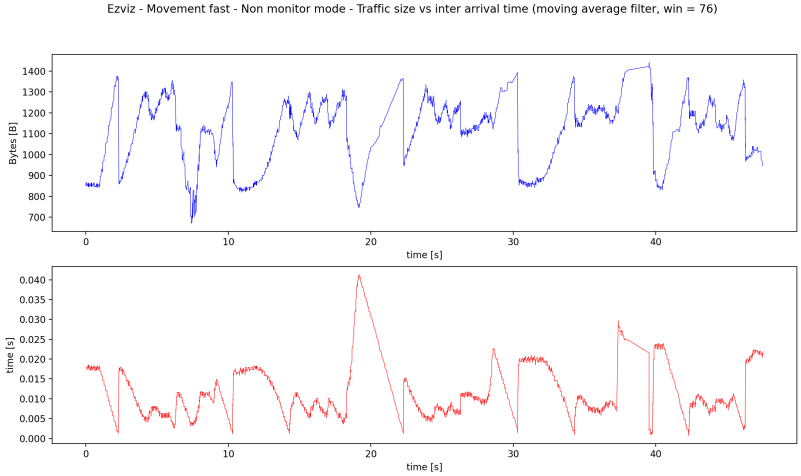

Figure 3.12: Ezviz - movement fast - non monitor mode

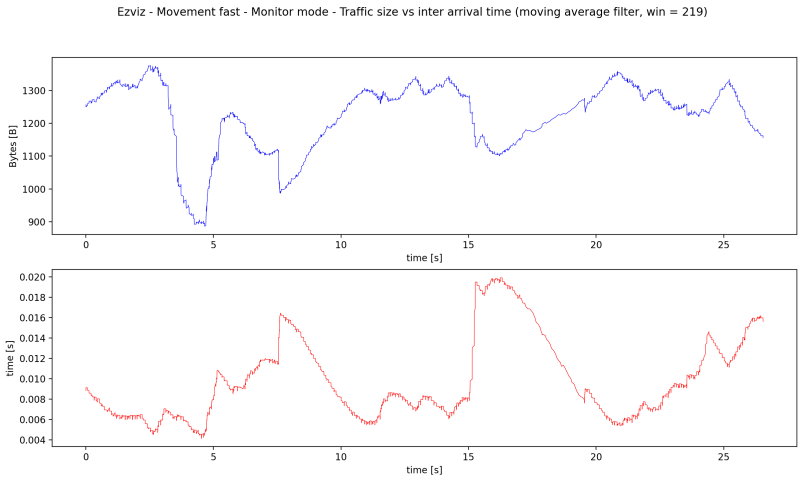

Figure 3.13: Ezviz - movement fast - monitor mode

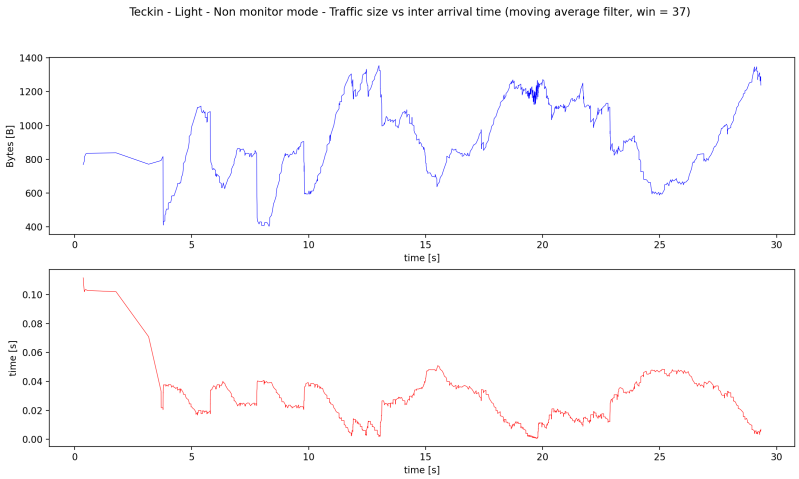

Figure 3.14: Teckin - light - non monitor mode

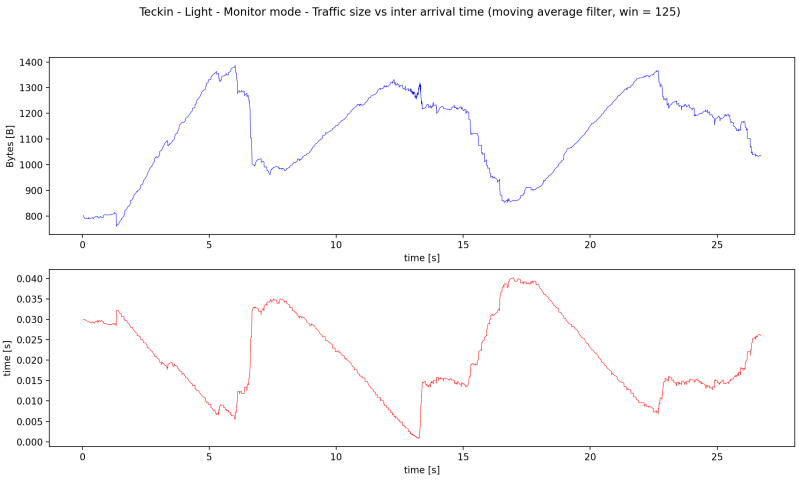

Figure 3.15: Teckin - light - monitor mode

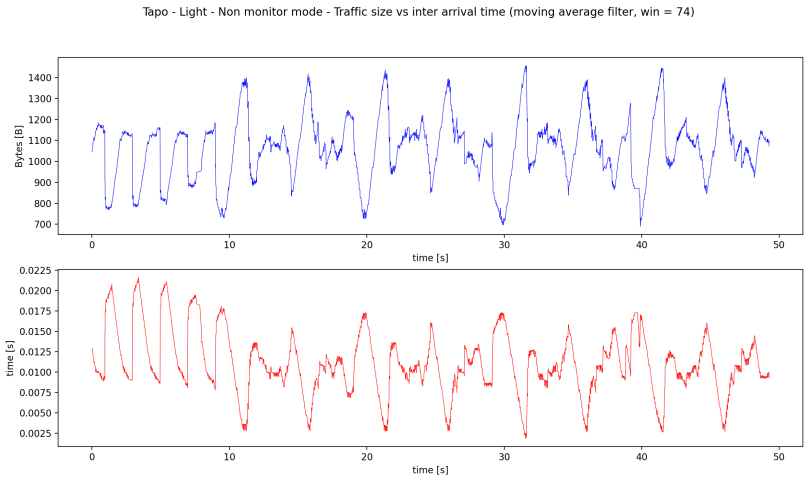

Figure 3.16: Tapo - light - non monitor mode

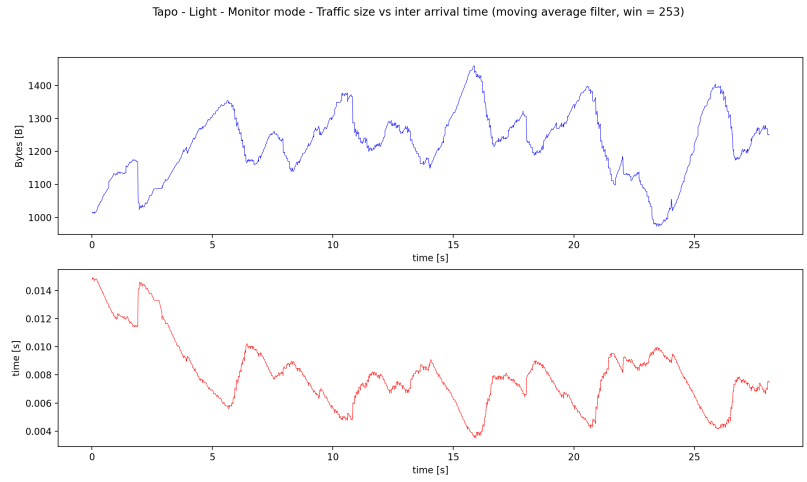

Figure 3.17: Tapo - light - monitor mode

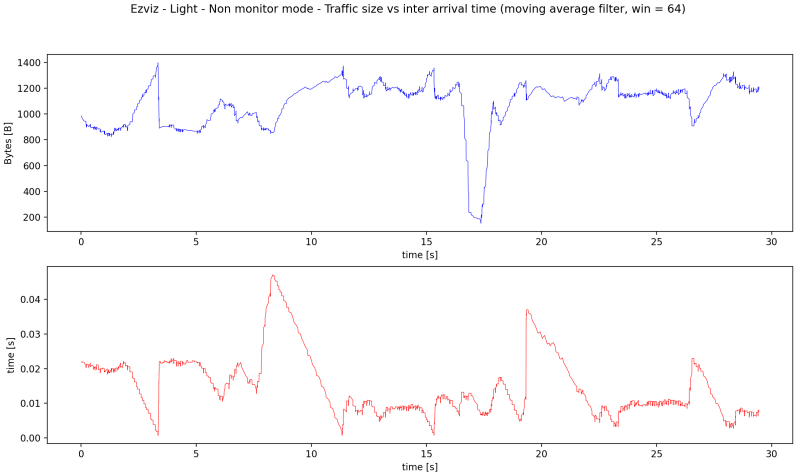

Figure 3.18: Ezviz - light - non monitor mode

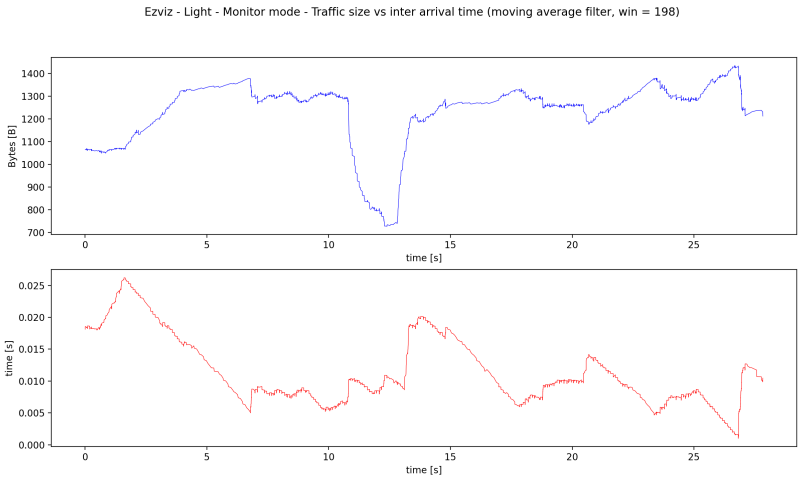

Figure 3.19: Ezviz - light - monitor mode

Chapter 4

Cameras specifications and system used

In this chapter, I provide a description of the main characteristics and features of the cameras that I have used for the whole work.
Also, I describe the system that I have programmed and built for capturing the data.

4.1 Teckin camera

The first camera is branded *Teckin*; the specific model is the *TC100*.
This is a Wi-Fi indoor camera with a resolution of 1080p (FHD). Its main feature are:

- **App control** The camera is fully usable through the Teckin app for smartphones and tablets.
- **Live view** Using the app, it's possible to have a live view of the FoV of the camera.
- **Night vision** Allows to have a better visibility in low light conditions.
- **Two way audio** The camera is provided with a microphone and a speaker, so that it's possible both to ear from the application the noises around the camera and to speak on the phone and ear from the camera what has been said. The usage of input/output audio can be enabled and disabled from the app.
- **Cloud storage** An additional service that allows the user to record videos and store them in the cloud.
- **Motion detecion** This is a very important feature for this work: it's the capability of the camera to understand if something is happening in its FoV. In case of movement, the camera sends a notification to the smartphone associated with it, so that the proprietary can check what's happening. Also, in case there is an active plan for the cloud storage, the video of what's happening can be recorded and stored to the cloud for playback.
  This feature can be activated and deactivated by the user; moreover, it's possible to tune the sensitivity of the camera to the movement using three levels (low, medium or high sensitivity; from low to high sensitivity, the camera tends to record videos for smaller movements); finally, it's also possible schedule when to keep this functionality active.

4.2 Tapo camera

Another camera I have used is the *TP-link Tapo C100*.
The resolution of this camera is again 1920x1080p (FHD). Other features:

- **App control** Possibility of controlling and setting the camera through an app.
- **Live view** Using the app, it's possible to check the view of the camera.
- **Night vision** Provides a visual distance of 30 ft (about 9 meters) in conditions of total darkness.
- **Two way audio** The camera comes with built in microphone and speaker that can both be enabled and disabled by the user.
- **Local storage** The camera is provided with a Micro-SD card slot for allowing local video storage.
- **Privacy mode** If enabled streaming and recording functions are temporarily disabled to protect the user's privacy.
- **Motion detection** This cam has the capability of understanding when there is movement in its FoV too. As for the Teckin camera, this feature can be enabled and disabled by the user whenever they want and the sensitivity of the camera to the movement is tunable (with the small difference that for this camera the tuning is continuous, not on only three levels). In case of enabled feature and movement detection, the camera sends a notification to the user so that they can check what's happening; a video can be recorded and stored on the Micro-SD (if present). Additionally, for this camera it's possible to choose the region of its FoV where it has to check for movements.

4.3 Ezviz camera

The last camera is the *Ezviz C6N*.
It has a resolution of 1920x1080p (FHD) and it comes with many features:

- **App control**
- **Live view**
- **Night vision** Up to 33 ft (about 10 meters).
- **Two way audio** Built in microphone and speaker.
- **Local storage** There's a slot for a Micro-SD card up to 258 GB.
- **Cloud storage** Possibility of activation of a cloud plan for video storage.
- **Sleep mode** When enabled, the device stops monitoring and recording.
- **Motion detection** This feature can be activated or deactivated by the user. The sensitivity of the camera to the movement can be tuned using seven levels. The area over which to detect movement can be adjusted too. In case of movement, a notification is sent to the user, a video is recorded and stored locally (if Micro-SD is installed) and/or on the cloud (if a cloud storage plan is active) and an audio warning is prompt by the camera (the intensity of the tone is tunable on three levels: silent, soft and intensive)

- **Pan and tilt rotation** The camera has a motorized pan and tilt for a 360° visual coverage. The rotation can be controlled by the user or automated, that is the camera uses its motion detection feature to follow any movement it detects.

4.4 One more device: Raspberry Pi

At this point, we have the cameras, but we need to connect them to a Wi-Fi router to have them ready to be used.
Actually, connecting them directly to the router may not be the optimal choice. In fact, in order to collect the data sent and received by the cameras at application level, we need to have a device able to capture the packets exchanged, and a commercial Wi-Fi router, which is not programmable, doesn't necessarily have such a feature.
For this reason, it's better to use a programmable piece of hardware that can be used as an access point (so it must have a Wi-Fi interface). The device that I have used is then a *Raspberry Pi 3 model B*, that is a small programmable computer provided with:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU.
- 1GB RAM.
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board.
- 100 Base Ethernet port.
- 40-pin extended GPIO.
- Four USB 2.0 ports.
- Four Pole stereo output and composite video port.
- Full size HDMI.
- CSI camera port for connecting a Raspberry Pi camera.
- DSI display port for connecting a Raspberry Pi touchscreen display.
- Micro SD port for loading your operating system and storing data.
- Micro USB power source up to 2.5A.

4.5 Programming the Raspberry Pi

The peripheral that have been used for the purposes of this work are the power supply port (of course), the Micro-SD slot, the Ethernet LAN port and the Wi-Fi interface.
The first thing that I have done is installing the operating system (OS) on the Micro-SD card. In order to do this, I have downloaded from the Raspberry website the version *2021-01-11-raspios-buster-armhf-full.img*, which contains the OS and some recommended files. Also, I have added a file with the name *ssh* and with no extension: its purpose is providing the raspberry with the instruction that are needed for letting the user connecting to the Raspberry itself using an ssh connection. At this point, the Micro-SD must be inserted into the Raspberry, which has to be powered on and connected to the router via LAN cable (this way, it gets an IP address by the router and the ssh connection can be established). Once found

the IP address of the Raspberry (which can be found on the configuration page of the router, reachable through the IP address of the router), we have to establish the connection, which can be done using the terminal and typing the command *ssh pi@<ip>*, where *ip* is the IP address of the Raspberry.
Now, the OS must be launched. This can be done by typing in the terminal window in which we have just established the connection with the Raspberry the following commands in this order:

1. *Sudo apt-get update*
2. *Sudo apt-get upgrade*
3. *Sudo shutdown -r now*

The last command is just for rebooting the Raspberry after having launched and upgraded the OS.

At this point, the Raspberry is active, but still it cannot act as an access point: to do this, we have to do some additional work on it.
Since the raspberry was shut down for rebooting, the ssh connection has teared down: first thing to do is establishing it again. At this point, we have to do the following:

1. Installing the hostapd access point software package:

   sudo apt install hostapd

2. Enable the wireless access point service and set it to start when the Raspberry Pi boots:

   2.1 *sudo systemctl unmask hostapd*

   2.2 *sudo systemctl enable hostapd*

3. Add a bridge network device (called *br0* in this case) by creating a file using the following command:

   sudo nano /etc/systemd/network/bridge-br0.netdev

4. Bridge the Ethernet network with the wireless network:

   4.1 Add the built in Ethernet interface (*eth0*) as a bridge member by creating the following file:

   sudo nano /etc/systemd/network/br0-member-eth0.network

   4.2 Enable the systemd-networkd service to create and populate the bridge when the Raspberry Pi boots:

   sudo systemctl enable systemd-networkd

5. Block the *eth0* and *wlan0* interfaces from being processed by the DHCP client on the Raspberry:

   5.1 Create the following file:

   sudo nano /etc/dhcpcd.conf

   5.2 Add the following line above the first line *interface xxx* (almost at the beginning):

denyinterfaces wlan0 eth0

5.3 Add the following line at the end of the file: *interface br0*

6. Ensure Wi-Fi radio is not blocked: *sudo rfkill unblock wlan*

7. Configure the access point software:

   7.1 Create the *hostapd* configuration file:

   sudo nano /etc/hostapd/hostapd.conf

   7.2 Add all the information needed to the configuration file just created (Figure 4.1 shows a sample of its structure). Apart from the trivial and free to choose information like the ssid and password of the Wi-Fi network, an important thing to set is the bandwidth of the network (if work at 2.4 GHz or 5 GHz) and the channel to use. For this work, I have chosen to use the 2.4 GHz bandwidth and the 7^{th} channel.

8. Restart the Raspberry to check that the wireless access point is automatically available:

   sudo systemctl reboot

```
country_code=GB
interface=wlan0
bridge=br0
ssid=NameOfNetwork
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=AardvarkBadgerHedgehog
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Figure 4.1: Sample of AP software configuration file for the Raspberry

At this point, it should be possible to connect to the Wi-Fi network of the Raspberry, which is a twin network of the one of the router to which the Raspberry is connected.

One last thing to do is installing the tools we'll need for capturing packets. The one I have chosen to use is *tcpdump*; to install it just run *sudo apt-get install tcpdump* on the terminal page on which we have previously established the ssh connection with the Raspberry.

4.6 Packet analyzer for PC

In order to collect data in monitor mode together with the capture on the Raspberry, the best thing is using a computer with a Wireless interface which is then able to capture packets from the surrounding environment.
In this case, the choice I took for sniffing packets is using the packet analyzer *Wireshark*. This program can be easily downloaded from the internet and then installed just by following the instructions. Once ready, since we want to use it for capturing

in monitor mode, we have to enable it. This can be easily done by opening the capture option window directly from inside the opening Wireshark page, find for the Wi-Fi interface and look for the column called *Monitor mode*: once there, just tick the option.
Another utility that is very useful to use is the *Airtool*. This tool is available for Mac computers and allows the user to choose to which Wi-Fi channel to be synchronized (both on the 2.4 GHz and the 5 GHz bands) on and on which Wi-Fi channel (again on both the bands) the user wants to capture packets.
Airtool is directly connected to Wireshark, meaning that when it's used for capturing packets, once chosen the channel on which to sniff, it automatically launches Wireshark as packet analyzer so that the captured packets are shown there for later analysis.

4.7 Mounting the system

Now we have all the pieces ready to mount the system for data collection.
What we have to do is simply turning on the Raspberry and connect it to the router via LAN cable (so to make the Raspberry Wi-Fi network work), turning on the cameras, follow the instructions on the apps to connect them to the WLAN of the Raspberrhy, and we are ready.
In fact, the cameras now work since connected to the internet and we can use directly the Raspberry to capture the packets on it. Additionally, we can also use a computer with installed a packet analyzer like *Wireshark* for capturing packets in monitor mode (so accessing up to the MAC level) while the Raspberry captures the same packets but up to the application level using the tcpdump tool. This way, we can analyze what is happening at both the levels and see if the results change depending on which information we have access to.
In Figure 4.2, there's a scheme of the whole system.

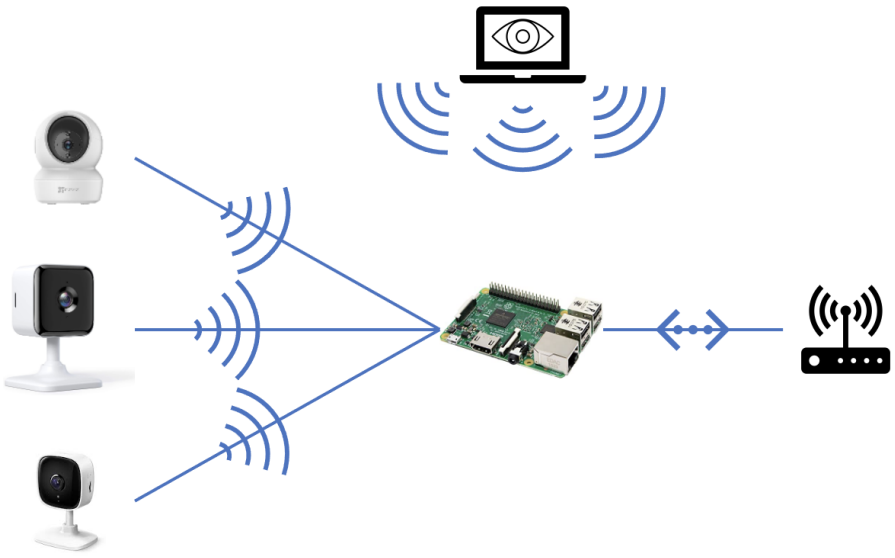

Figure 4.2: Scheme of the system for data collection

Chapter 5

Data collection and feature extraction

At this point, we have the system ready for collecting the data we need for the purposes of this work. What we have to do is (in this order):

1. Deciding which activities we want to try to identify.
2. Collecting the data.
3. Extract the feature from the data.

5.1 Activities to be identified

The first thing to do is deciding what we would like to understand about what's happening in the FoV of a camera.
The literature suggests that it should be possible to distinguish among several activities like walking around, sporting, hairstyling, turning on/off a light, eating, watching TV and a lot more. Anyway, those studies were performed with a lot of volunteers that were available to actually do such activities in front of a cameras for very many times. In this work, for practical reasons, there was a single volunteer (me) available for collecting the data that were needed, so the number of activities I have decided to try to distinguish among are four:

- **No activity** Simply, nothing is happening in the FoV of a camera.
- **Movement slow** Someone is moving quietly and slowly in front of the camera. This is like a normal situation in which someone is walking around normally in a room.
- **Movement fast** Now, there still is someone moving around in the FoV of a camera, but their movement is more frenetic, as if they were in a hurry or trying to do something suspicious.
- **Light event** A light is turned on or off in the room where the camera is placed.

The choice of these four activities was of course also dictated by the starting point of this thesis: the forensic interest. Indeed, even if they are not so specific, they can be used to reconstruct (with a certain level of accuracy) the events happening in the room where the camera is placed and therefore to prove or disprove a thesis.

For example: its more likely for a thief entering a house to move more frenetically than for the proprietary. So, a situation in which we have a sequence of *Movement fast* events alternated with *No activity* events may be used to support the thesis that someone was committing some kind of crime (like a robbery). Vice versa, a person moving quietly is more likely that they are not doing nothing suspicious.

5.2 Data collection

Of course, in order to train the classifiers and test their accuracy, we need a lot of data, that must be collected.
In order to do this, the first thing to do is setting up the system, which means turning on the Raspberry, connecting it to the router and power on the cameras.
At this point, we have to connect to the Raspberry (that is, opening an ssh connection with it) and prepare the command to be used for capturing packets. Since I have chosen to used tcpdump as tool, the command is the following:

sudo tcpdump -i wlan0 -w <path_to_folder>:

The syntax is the following:

- The key work *sudo* is used to lauch the command *tcpdump* with root privileges.
- The option *-i* is used to choose the interface on which to sniff. It must be followed by the name of the interface, that is *wlan0* in this case.
- The option *-w* is used to specify that we want the captured file to be saved at a specific location in the file system; where is specified in the *<path_to_folder>* parameters, whose last part must be the name of the file to be saved with its extension. In this case, the extension is *.pcapng*, that is readable by Wireshark as a packet captures for later analysis.

Once the command is ready, we have to launch it, but still we cannot start performing activities and collecting data: before, we have to use Airtool for capturing also in monitor mode.
To do this, we must know the channel on which the cameras are transmitting. This is an easy task since we have programmed the Raspberry to work on the 2.4 GHz bandwidth and on the 7^{th} channel, so we just have to choose these parameters. In case we do not have these information, there are two options: the first is trying more channels until when we don't find the correct one, the second is using the Airtool option for switching periodically the channel on which to capture. The second option is clearly faster, but it also gives us less information since only once in a while we will capture from the single channel we are really interested in. Therefore, even if more time consuming, it would be better to spend some time looking for the channel on which a camera is transmitting, provided that we have enough time to make attempts to retrieve this information (in fact, if the camera doesn't transmit for enough time, we may not find the channel within the time the transmission lasts).
Once we have retrieved also the information on the Wi-Fi channel used, we are ready: we launch tcpdump on the Raspberry, then we go into Airtool and choose the channel on which to capture and, once Wireshark is opened, we can proceed performing activities in front of the camera.
For each activity, the data collection consisted in spending 30 minutes performing that activity, so:

- **No activity** Just let the camera frame the same still image for half an hour.

- **Movement slow** Walking slowly (so, just walking) in the FoV of the camera.
- **Movement fast** Running or doing some unusual and frenetic movement in front of the camera.
- **Light event** Turning on off the light of the room in which the camera is placed (that is completely dark when the light is off) at regular interval of 2 seconds (the reason of the choice of this interval will be clear when I will explain how features have been extracted from the captures).

An important observation that needs to be done is that these data have been captured with the audio on the cameras turned off (both in input and output), so to capture information on the video traffic only. This is very important to keep in mind since (later in this work), I will describe what changes if audio is turned on.
When the 30 minutes expired, the captures had to be stopped, which means first of all stopping the Wireshark capture on the computer, save it and then stop the tcpdump command on the Raspberry.
Note that the order in which these commands (both at the beginning and at the end of the data collection phase) is not invertible. In fact, when we launch the monitor mode capture on the computer, its Wi-Fi interface cannot anymore used to send commands since it's busy at looking at the Wi-Fi channel, which means we cannot launch tcpdump on the Raspberry. The same holds for the end of the captures: if we don't stop the capture on the computer, its Wi-Fi interface cannot exit the monitor mode and therefore it wouldn't be possible to stop the tcpdump command on the Raspberry.
One last thing to do is, since the feature extraction happens all on the computer, sending the captures made on the Raspberry to the computer itself. This can be easily done using the *scp* (which stands for *secure copy*) command on the terminal. Just open another terminal window and type:

scp pi@<raspberry_ip_address>: <path_to_file_on_remote_client>
<path_to_file_on_computer>

Once captured the data for each one of the four activity both in monitor mode (on the computer) and non monitor mode (on the Raspberry) for all the three cameras, we are ready for extracting features from them.

5.3 Choice of the features

At this point, before moving to the feature extraction procedure, we have to define the statistics we may be interested in for our purposes.
Remember that the starting point of this analysis was the information leakage as a result of the difference coding techniques for encoding video traffic: depending on how many different pixels there are between two consecutive frame (or more in general between the I frame of a GoP and each the P and/or B frames of the same GoP), the packet corresponding to a frame will be longer or shorter. Moreover, as we can see from the graph in the Section 3.5, also the inter-arrival time is subject to some modifications in correspondence of the instants in which there is an event in the FoV of a camera. Finally, if we put together the information on the packet size and on the inter-arrival time, we can also derive information on how the rate at which packets are sent/received is different.
All of this suggests that the statistics we have to define are related to the packet size, the inter-arrival time and the rate. Here follows Table 5.1, in which there is the list of all the features that are used and a brief description of it; a more detailed

description of some of the features will also follow the table. (Note that for uplink we mean the packets sent by the camera and with downlin the ones received by it).

Activity	The label of the activity corresponding to the current capture file
Avg_UL_rate	Average rate for uplink packets
Avg_DL_rate	Average rate for downlink packets
Ratio_avg_UL_DL_rate	Ratio between the average uplink and downlink rate
Var_UL_rate	Variance of the uplink rate
Var_DL_rate	Variance of the downlink rate
Skew_UL_rate	Skewness of the uplink rate
Skew_DL_rate	Skewness of the downlink rate
Kurt_UL_rate	Kurtosis of the uplink rate
Kurt_DL_rate	Kurtosis of the downlink rate
Avg_UL_iat	Average inter-arrival time for uplink packets
Avg_DL_iat	Average inter-arrival time for downlink packets
Var_UL_iat	Variance of the uplink inter-arrival time
Var_DL_iat	Variance of the downlink inter-arrival time
Skew_UL_iat	Skewness of the uplink inter-arrival time
Skew_DL_iat	Skewness of the downlink inter-arrival time
Kurt_UL_iat	Kurtosis of the uplink inter-arrival time
Kurt_DL_iat	Kurtosis of the downlink inter-arrival time
Avg_UL_pkt_len	Average uplink packet length
Avg_DL_pkt_len	Average downlink packet length
Var_UL_pkt_len	Variance of the uplink packet length
Var_DL_pkt_len	Variance of the downlink packet length
Skew_UL_pkt_len	Skewness of the uplink packet length
Skew_DL_pkt_len	Skewness of the downlink packet length
Kurt_UL_pkt_len	Kurtosis of the uplink packet length
Kurt_DL_pkt_len	Kurtosis of the downlink packet length
Num_UL_pkt	Number of uplink packets sent
Num_DL_pkt	Number of downlink packets sent
Pdf_UL_pkt_len	Distribution of uplink packet length
Pdf_DL_pkt_len	Distribution of downlink packet length

Table 5.1: Table of features used

As we can see, the statistics can be divided indeed into three groups: the one related to the rate, the ones related to the inter-arrival time and the ones related to the packet length. For each one of these three categories, I have computed the average, the variance, the skewness and the kurtosis.
Some of the features deserves some additional comment:

- **Average UL/DL rate** These rates have been computed as the average of the instantaneous rate, which in turn has been computed as the ratio between the length of a packet and the time intervening between when the previous packet had been received and the time at which the current packet has arrived.
  A countermeasure had to be taken for packets whose inter-arrival time was shorter than one microsecond (that is the time resolution of Wireshark). In fact, this situation would result in having two packets with the same arrival time, so inter arrival time equal to zero and infinite rate, which has no physical sense. Therefore, to avoid this error, if the inter-arrival time was zero, it was forced to be one microsecond.

- **UL and DL packet length distribution** These two features consist in a list in which the i^{th} element contains the number of packets whose length is

between $100*(i-1)$ and $100*i$ bytes, with the last element of the interval (that is exactly $100*i$) excluded. Each list has 17 elements; in fact, I have noticed that it's common to have packets whose length is at most 1700 bytes and only very rarely there was a longer packet. For this reason, a packet longer that the threshold 1700 bytes was considered as an outlier and skipped.

5.4 Time window for feature extraction

Now, what has to be defined is the time window over which the features corresponding to a certain capture file are extracted.
The value of the time window I have chosen is the result of two observations:

1. We have understand that the I frames are useless for our purposes since what we are interested in are the differences between two frames of the video, and this information is encoded into the P and B frames. Therefore, the influence of one of the I frames (since we have noticed that there could be more than one) of each GoP must be cancelled. This can be done by considering a time window in which there is always at least one I frame and by summing the volume of all the packets in that time window.

2. By looking at the graph in Section 3.5, we can notice that there is always a peak about every 2 seconds.

Therefore, 2 seconds is the time window I will consider for extracting the features.

5.5 Feature extraction procedure

The feature extraction procedure consists in the scan of all the packets of each capture in order to compute some statistics on the traffic exchanged between the camera and the access point (so the Raspberry).
The procedure is similar for the two modes, monitor and non monitor; only the filters used to detect the packets to be considered are different (since at MAC layer and application layer the information that we have are quite different).
First of all, the program performing feature extraction is written in *Python* programming language, since it provides nice and easy to use libraries both for analyzing a .pcapng capture file and for storing the features regarding a capture in a .csv file. Those libraries (respectively the *pyshark* and *pandas* libraries) have to imported at the beginning of the program.
Then, the feature extraction procedure assumes that we know the MAC addresses of the cameras that we are considering. Indeed, we are still in the training phase, so taking this assumption is reasonable. The know MAC addresses of all the cams are stored in a list that will be called *MAC_CAMS*.
At this point, we have to declare the variables that are needed. For each one of the three cameras, I have declared the following variables:

- A dictionary for storing the features. Actually, each one of this variable is handled as a nested dictionary in which the outer key (that is the key identifying a dictionary within the nested dictionary) is the index of the window over which the features have been computer, while the inner keys are the names of the features (of course, each one has its value assigned).

- Two lists, one for the uplink and one for the downlink, in which the i^{th} element is the time at which the i^{th} packet was sent/received.

- Two lists, one for the uplink and one for the downlink, in which the i^{th} element is the length of the i^{th} packet sent/received.
- Two lists, one for the uplink and one for the downlink, in which the i^{th} element is the inter-arrival time of the i^{th} packet sent/received.

Also, I need a variable that work as index for understanding when we are changing window for computing the features over the current window itself; I'll call it *win_index*. Now, what we have to do is opening the capture file (which is done by storing in a variable the result of a call to the function *pyshark.FileCapture* which in turn needs as argument the path to the capture file to open) and scan one by one all the packets belonging to it.
Now, we need to consider the possibility that a captured packet may be malformed and therefor unreadable. To avoid errors each time this happens, I have used a *try-except* statement: in case some error was generated because of the presence of a malformed packet, the *except* command just impose to go to next packet using a *continue* command.
At this point, we need to distinguish between the procedure used in monitor mode and in non monitor mode.

- **Non monitor mode** In this case, we have access to all the layers of each packet up to the application level. Of course, we are only interested in data packets (since video traffic is generated at application layer from the cameras), so what we have to do is looking for the packets sent/received by the MAC address of one of the cameras and that comes from the application layer. The procedure is then, for each non malformed packet:

  1. Extract source and destination MAC address.
  2. Extract the highest layer of the packet.
  3. Check if the source/destination MAC address belongs the the list of known MACs and that its highest layer is one among the following: DATA, HTTP, HTTPS, TLS, TCP, UDP, _WS.MALFORMED. These protocols are the ones that have been identified by inspection directly on the capture file to be packets generated or received by the cameras. Note that I have included also the _WS.MALFORMED protocol. Actually, this stands for packets whose application layer data is malformed, not for completely malformed packets which are discarded at the beginning. The reason why this protocol is included is that what we care is that, even if application data is malformed, the information on the length of that data is still present and consistent, and since the length it's all what we care (together with the arrival time), we are good like this.

  If a packet is sent/received by one of the cameras and its a data packet, then we have to check if its arrival time is within the current time window of 2 seconds or not.

  1. If it's within the window, then we just have to update the lists corresponding to the traffic size (UL if the packet is sent by the cam, DL if it's received by it), and the arrival time (same distinction between UL and DL) so to compute the inter-arrival time and update the corresponding list. For the volume of a packet, I have chosen to consider its length as the sum of the length of the IP header, the transport layer header and the data layer payload; the reason is that in this way we have consistent information with the analysis performed in monitor mode since what we are doing is equivalent ti consider the length of the payload at MAC level.

2. If the arrival time of the packet is such that it belongs to the next window, then we have to use the lists corresponding to the UL and DL traffic size and inter-arrival time to compute the features for the window that we are leaving. Once computed, the lists for the previous window are cleared and updated with the traffic size and inter-arrival time of the last packet, the window index is increased and the process restarts. This until when all the packets of the capture file have been analyzed.

- **Monitor mode** Now, we have the information only up to the MAC layer. Again, we are interested in the data packets sent or received by one of the cameras, but the packets that are identified as data packets at MAC layer may be control packets coming from the IP or transport layer. For this reason, I only consider the data packets whose length is above a certain threshold that I've set to 28 bytes. This is a quite low threshold, but it's the one that guarantees that no short data packet (at application layer) is lost; in fact, in order to come from the application layer, a packet must be long at least as the IPv4 header (that is 20 bytes) and the shorter header between the UDP and TCP one, so as the UDP one (that is 8 bytes).
  At this point, what we have to do is the following:

  1. At MAC layer, we may have up to four addresses (source, transmitter, destination and receiver address) for each packet. What we want is considering as packets sent by a camera the ones whose source or transmitter address is the one of the camera we know it's transmitting and as packets received by the camera the ones whose destination or receiver address is the one of the camera. So what we have to do is trying to extract for each packet all of the addresses it has in its header (note that not all of these addresses are always present in a packet).
  2. Now, we have to check if the source or the transmitter or the destination or the receiver address belongs to the *MAC_CAMS* (note: actually, we only check if the valid addresses belong to the *MAC_CAMS* list).
  3. Also, the packet must be a data packet. This means that we have to extract its *type* and *subtype* values. The values in these fields are encoded as numbers, in particular the data packets are all the ones that have $type = 2$, while for the *subtype* it has to be $0 \leq subtype \leq 3$ or $8 \leq subtype \leq 11$. If the two conditions are both true, then we have found a data (or QoS data) packet.
  4. Finally, the length of the payload must be at least long 28 bytes (for the reasons explained above).

  Once we have found a data packet, the procedure is exactly the same followed in the non monitor mode: check the arrival time of the packet and depending on if belongs to the current window or to the next one just update the lists or compute the statistics.

Some important observations have to be done:

- For each camera and both in monitor and non monitor mode we have considered the data packets sent and received by a camera. Note that, since video traffic is only exiting the cameras, the traffic incoming in a camera is almost always consisting in ACKs. This is true both for TCP or TLS packets, but also for UDP ones (in this case, the ACKs are generated at application level and sent via UDP protocol). This means that in general the uplink direction is the one in which we will observe more variance of the features.

- It may happen that within a window there are no data packets (regardless of the mode of the analysis). In this case, a natural computation the features is impossible, just because the lists that should be used are empty. The countermeasure I have adopted is using a dummy value to identify windows in which there was no data sent; this value is -1. So, each time a feature has a value equal to -1, it means that there was no sufficient data for computing that feature over a certain window; therefore, that window must not be considered by the classifiers.

- Note that, for how we have considered the volume of the packets in non monitor mode, the information on the traffic size of the packets is the same for both the modes. But still, the actual knowledge that we have in monitor mode is less than the one we have in non monitor mode; indeed, in monitor mode we are for sure considering the packets that we are considering in non monitor mode, but also a lot of other packets that at MAC layer appears as being data packets but only because they come from one of the upper layers (so also IP and transport layer). So, in non monitor mode we have more guarantees on the nature of the packets we are analyzing.

- Sometimes, some packets appeared as badly encoded, that is, the program scanning the capture file is not able to decode a packet and read it, generating an error that makes the program stop in any case (the packet is not read at all, meaning that it's not identified as malformed and therefore it's not skipped but it stops the execution of the program). When this happens, the only solution is opening the capture file and filtering out the packet that has generated the error; this has to be done for all the packets that may cause such an event.

5.6 Saving the dataset

Once the feature extraction procedure is finished, that is, the packets of a capture have all being scanned, we have to convert the dictionaries in which the features have been stored into datasets and store them.
The data structure that I have chosen for saving the datasets is the *Pandas Dataframe*: a nested dictionary like the ones we have built can be easily transformed into such a data structure by using the function *pandas.DataFrame*, which takes as input the dictionary and gives as output a pandas dataframe. Actually, we also have to transpose the resulting dataset (otherwise its lines would be the features and its columns the window index, but we want the exact contrary).
Once obtained the pandas dataframe, we just have to save it as a .csv file. This is done by applying the function *to_scv* to the dataset that we have just created; the argument it needs is the path to the folder where we want to store the dataset (the last part of the path must be the name of the file with the .csv extension).

Chapter 6

Camera and activity classification: results

Now, we have all the datasets ready for the classification phase.
In this Chapter, I will describe how data have been preprocessed, the classifiers I have used for camera and activity identification and the results I have obtained.

6.1 Environment for data classification

For this part of the work, I have used *Jupyter lab*. The reason of the choice is that this framework provides a nice and easy way of using Python notebooks (*.ipynb)* files), which are very useful in this case since they allows to run pieces of code separately and looking at the results as soon as they are ready rather than all the end of the processing.
To launch Jupyter lab, just open a terminal window in the folder where the files to be used are located and then use the command *Jupyter lab*: this will open like an internet page, but it's a local one in which it's possible to select the kernel we want to use (Python in this case).

6.2 Data preprocessing

There are some common operations that have to be done on all the datasets in order to have them ready for the classification phase.

1. The first thing thing we need to do is removing all the rows that contains inconsistent values. Remembering how we have coded the presence of meaningless value during the dataset preparation phase (that is, the value of the feature having no sense is -1), we just have to check that for each row and for each column there are no values equal to -1; in case there is one or more, we just delete the corresponding row.
   This can be easily done just by using the *drop* function on the variable in which the dataframe containing the data is stored.
   At this point, there are only meaningful values in our dataset.

2. The next thing to do is removing the meaningless columns that may have been created by the program when the nested dictionary containing the features had been transformed into a pandas dataframe. In case there are some, these ones simply contains primary keys corresponding to the window index we

have used, they are located on the left side of the dataset and they are called *Unnamed: 0*, *Unnamed: 0.1* and so on.

3. Finally, there is another important thing to adjust, and it's relative to the features describing the packet length distribution in uplink and downlink. The fact is that these are read as list (actually, this is not automatic; how to make this happen will be described later on) when we open the dataset, but the classifiers are not made for working with features that are list. So, we have to transform these lists so to obtain a single feature for each element of the list. The procedure to do this is following;

   - Create as many supplementary lists as the number of elements of the packet length distribution features (so 34 lists, 17 for the uplink and 17 for the downlink). The i^{th} list for the uplink/downlink will contain all the i^{th} element of the packet length distribution feature for all the rows.
   - Loop on all the rows of the datasets and consider the packet length distribution feature: for each row, we have to place in the i^{th} element of the feature in UL/DL in the i^{th} list of the UL/DL.
   - At the end of the process, the i^{th} supplementary list for the UL/DL will contain the i^{th} element of the UL/DL packet length distribution of all the rows of the dataset.
   - Create a new feature for each supplementary list and remove the old UL/DL packet length distribution features.

4. Finally, we have to merge the single datasets; which ones have to be merged depends on the classification that is being performed, so this information will be given later on, directly in the sections of interest.

6.3 Regarding the results

At this point, everything is ready for proceeding with the classification procedures. Before going on, I want to give a brief description of what will follow, so that results can be checked more directly:

- Section 6.4 describes the general structure of the classification procedure, so how the train and test procedure is performed.
- Section 6.5 describes the classifiers I have tested in the experiments.
- Section 6.6 describes the metrics that have been used to measure the performances of the classification.
- Section 6.7 describes the extra preprocessing needed for performing the classification of the camera that is transmitting and the performance of the classification both in non monitor and monitor mode.
- Section 6.9 describes the performance of the tested classifiers for the single cameras and for both the non monitor (Sections 6.9.1, 6.9.3 and 6.9.5) and monitor (Sections 6.9.2, 6.9.4 and 6.9.6) modes.
- Sections 6.11 reports the performance of the classification by using a single big classifier for all the cameras.
- Section 6.12 consists in the reportage of the results of the classification procedure with the audio feature of the cameras activated both in input and output.

6.4 Structure of the classification procedure

The train and test procedure is performed by using a stratified 10-fold cross-validation. This means that the dataset is split in 10 portions and that the the train and test is performed 10 times: each time, nine of the parts in which we have split the dataset are used for to train the model and the remaining one is used for the test; of course, the test set changes each time, and the final performances are provided as an average of the ten attempts.
The train and test sets for the cross-validation procedure are created using the *train_test_split* function from the *sklearn* python library. This needs as arguments:

- **The target variable** It's the feature target of the classification, the one we want to check wether it can be guessed or not and with which accuracy. This variable is the name of the camera for the camera classification procedures and the activity for the activity classification procedures.
- **The input variables** The attributes used to guess the target one, so all the ones that are not the target one (all the features extracted from the captures).
- **The test set size** The dimension of the test set with respect to the dimension of the whole dataset, so it must be a number between 0 and 1. In my experiments, I have chosen to set this number to 1/3.
- **A random state** This is an integer seed used for obtaining reproducible results.
- **A shuffle flag** If set to *True*, the dataset is shuffled before splitting.

The outputs of the function are sets used for training and testing the classifiers.
At this point, we are ready to test the performances of different classifiers.

6.5 Classifiers used

In order to have a consistent idea of the level of precision that can be obtained, I have tried to use a lot of classifiers.
What is following is a list of all the ones I have tried with the parameters I have used to set them:

- **Naive-Bayes classifier** I have used the Gaussian Naive-Bayes variant: this follows a normal distribution, so to support continuous data too.
- **K-NN classifier** The K parameter has been set to 5, so the classifier classifies an example though majority voting among the labels of the 5 nearest neighbors of the example itself. The nearest neighbors are found by using the *KDTree algorithm*.
- **Decision tree classifier** Here, the criterion according to which a new branch is added to the tree is the best split according to the Gini index parameter. Performances have then been tested using different *max_depth* for the tree. In the results, I report the performances using *max_depth*=3 for the activity classifier and *max_depth=None* for the camera classifier.
- **Bagging classifier** I have tried to use this ensemble classifier in two different flavours and in both the cases with the parameters *n_nestimators*=50 (so 50 base classfiers of two different types):
    1. Using Decision Tree classifiers with *max_depth*=3

2. Using K-NN classifiers with K=10 and KDTree as neighbor computation algorithm.

- **Random forest classifier** Again, I have used 50 Decision Tree classifiers base, each one with *max_depth*=3 and using the paramet *oob_score*=*True*, so the performance are evaluated using the out of the bag score.

- **Extremely randomized tree classifier** Just like the Random Forest classifiers, with the difference that bootstrapping is not performed and the split is randomized at each level of the tree. Parameters are *max_depth*=3 and *n_nestimators*=50.

- **Adaboost classifier** Application of the Adaboost procedure to 50 base Decision Tree classifiers, each one with *max_depth*=3

- **Support Vector machine classifier** All parameters are left as the default ones; the most important is the *decision_function_shape* parameter, which is set by default on the value *ovr*, so the approach is one-vs-rest.

6.6 Classification metrics

Here, we are dealing with a multi-label classification procedure. Indeed, the classifiers have to distinguish among three cameras (for the camera classification procedure) and four activities for each camera.
In this cases, the best way of evaluating the performance of a classifier is by passing though the confusion matrices. In order to explain how a confusion matrix is built, let us suppose we're dealing with a binary classification problems in which the two classes a classifier has to distinguish are *Yes* and *No*: in this case, a confusion matrix is a 2x2 matrix on which we find on the rows the true classes and on the columns the predicted ones. For a better understanding of how a confusion matrix work, consider Figure 6.1: The values in its entries are:

- **TP (True Positive)** is the number (or the percentage) of examples whose true class is *Yes* that have been classified as *Yes*.

- **FP (False Positive)** is the number (or the percentage) of examples whose true class is *No* that have been classified as *Yes*.

- **FN (False Negative)** is the number (or the percentage) of examples whose true class is *Yes* that have been classified as *No*.

- **TN (True Negative)** is the number (or the percentage) of examples whose true class is *No* that have been classified as *No*.

Sticking to a binary classification problem, the main metrics that can be computed by looking at the confusion matrix are:

- **Accuracy** Returns the ratio between the number of examples correctly classified (TP and TN) and the total number of examples, so in equation:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{6.1}$$

Therefore, this is a measure of how many examples the classifiers have correctly classified with respect to the total number of examples.

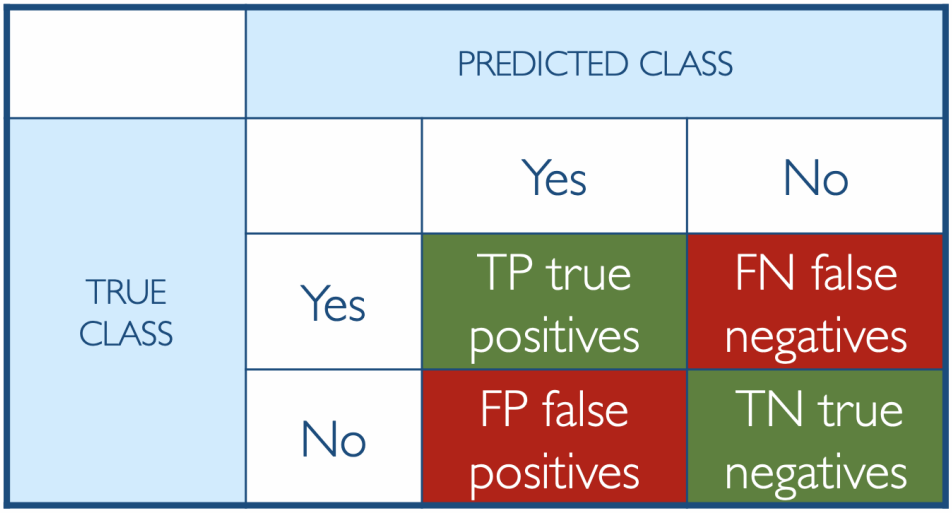

Figure 6.1: General structure of a confusion matrix

- **Precision** It's the number of positive examples (class *Yes*) that have been correctly classified with respect to the total number of examples classified as positive:

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

- **Recall** It's the number of positive examples (class *Yes*) that have been correctly classified with respect to the total real number of positive examples:

$$Precision = \frac{TP}{TP + FN} \tag{6.3}$$

- **F1-score** This is an hybrid metric that keeps into account both the precision and the recall metrics by making a sort of geometrical average between them:

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6.4}$$

The structure of a confusion matrix can be simply extended to a multi-label classification problem (so one in which there are more than two labels the classifiers have to distinguish). If we suppose that the general class i occupies the i^{th} row (the true class) and the i^{th} column (the predicted class) of the matrix, we have that:

- The numbers on the diagonal of the matrix (so the ones in positions $[i, i]$) are the absolute number (or the percentage) of the elements of class i correctly classified (so as belonging to class i).
- The number in the general position $[i, j]$ contains the absolute number (or the percentage) of examples whose real class is i but that have been classified as class j (so committing a classification error).

By considering this generalization for the confusion matrix, we can easily understand also how the metrics described above can be generalized and used to understand how good a classifier is for our multi-label camera and activity classification problems.

6.7 Camera classification

As we have noticed in Section 3.5, different cameras produce different traffic size and/or inter-arrival time patterns for the the same activity. Therefore, it's interesting for the purposes of this work understanding if it is possible using the same features that will be used to perform activity identification to perform also camera classification.
In order to do this, we first need some further preprocessing:

- Since we want to use all the data we have to try to identify a camera, we have to consider the datasets corresponding to the four different activities for each camera and merge them, so to obtain 3 datasets, one for each camera.

- In order to distinguish among cameras, a classifiers needs an example to be provided with a feature that says from which camera this example has been collected. So, for each of the three datasets just built, we have to add a column (that is, a feature) in which it's written the brand of the camera.

- For the purpose of classifying a camera, the activity feature related to each example is useless, so it can be dropped.

At this point, the data is ready to be used as input for the classifiers.
In the following Sections 6.7.1 and 6.7.2, I report the confusion matrices resulting from the camera classification of the different classifiers used with the related performance metrics respectively in non monitor and monitor mode.

6.7.1 Non monitor mode

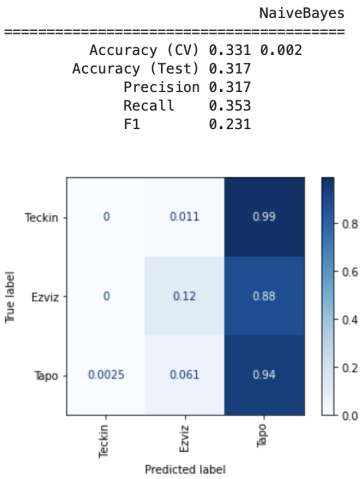

Figure 6.2: Camera classification - Non monitor mode - Naive Bayes classifier

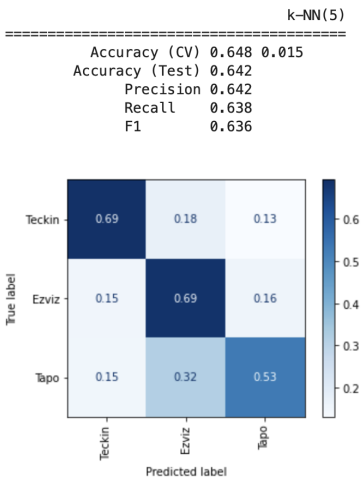

Figure 6.3: Camera classification - Non monitor mode - K-NN classifier

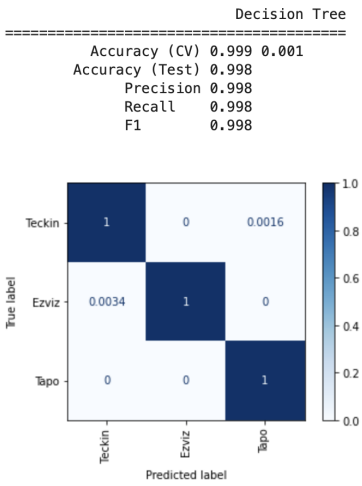

Figure 6.4: Camera classification - Non monitor mode - Decision Tree classifier

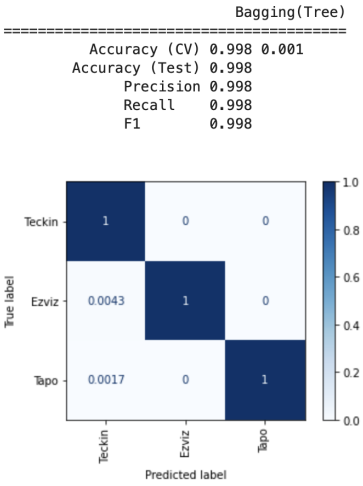

Figure 6.5: Camera classification - Non monitor mode - Bagging (tree) classifier

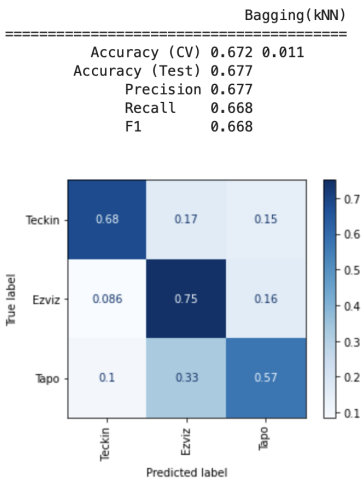

Figure 6.6: Camera classification - Non monitor mode - Bagging (KNN) classifier

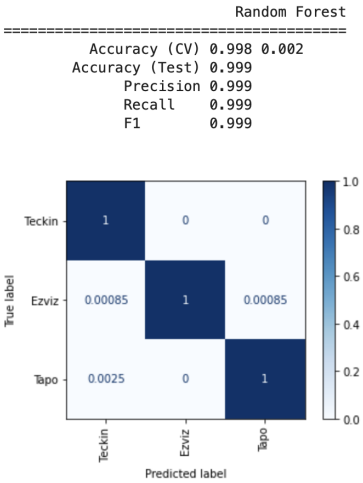

Figure 6.7: Camera classification - Non monitor mode - Random Forest classifier

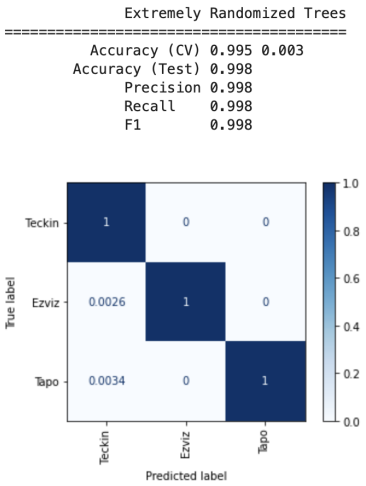

Figure 6.8: Camera classification - Non monitor mode - Extremely Randomized Trees classifier

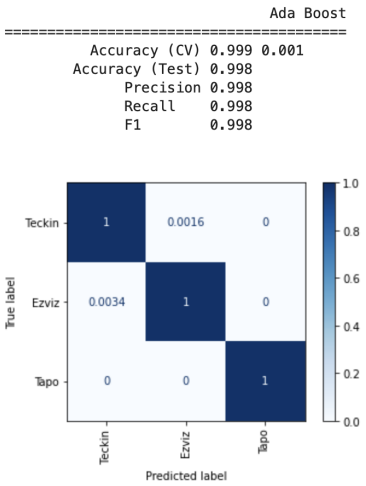

Figure 6.9: Camera classification - Non monitor mode - Adaboost classifier

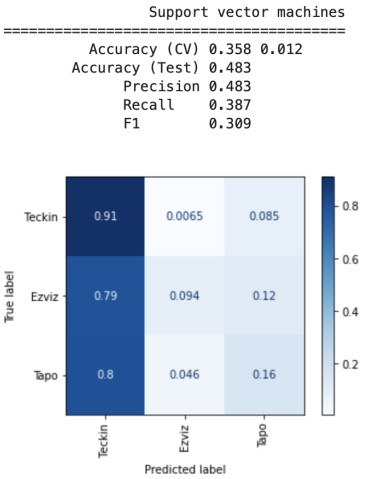

Figure 6.10: Camera classification - Non monitor mode - Support Vector Machine classifier

6.7.2 Monitor mode

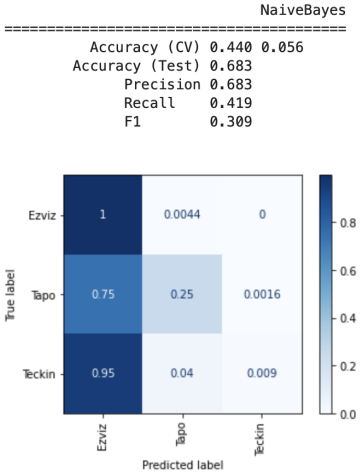

Figure 6.11: Camera classification - Monitor mode - Naive Bayes classifier

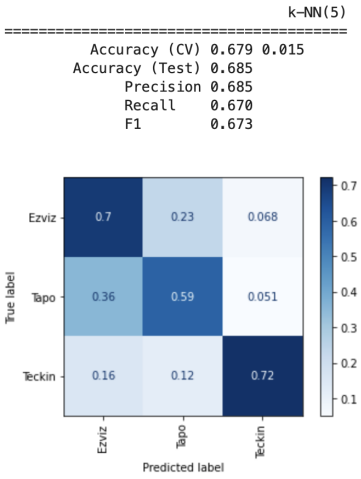

Figure 6.12: Camera classification - Monitor mode - K-NN classifier

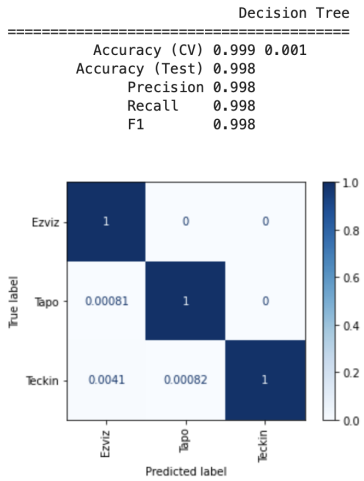

Figure 6.13: Camera classification - Monitor mode - Decision Tree classifier

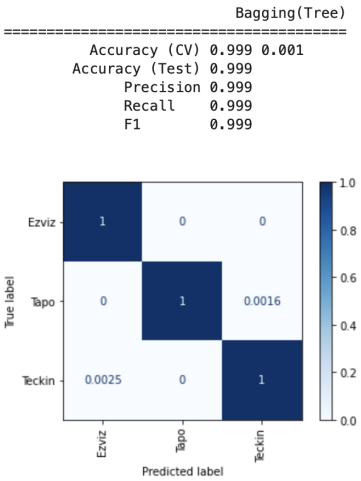

Figure 6.14: Camera classification - Monitor mode - Bagging (tree) classifier

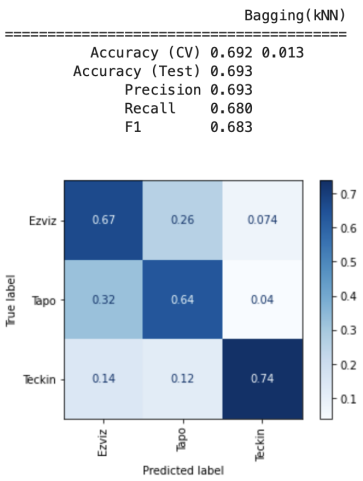

Figure 6.15: Camera classification - Monitor mode - Bagging (KNN) classifier

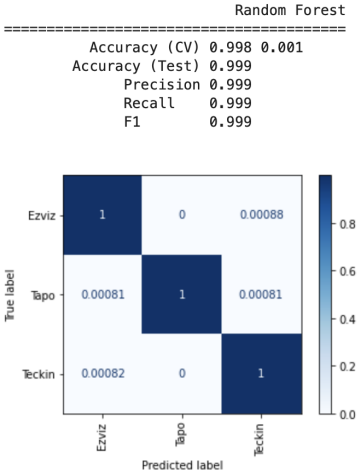

Figure 6.16: Camera classification - Monitor mode - Random Forest classifier

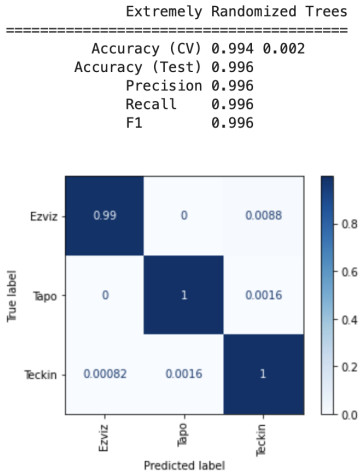

Figure 6.17: Camera classification - Monitor mode - Extremely Randomized Trees classifier

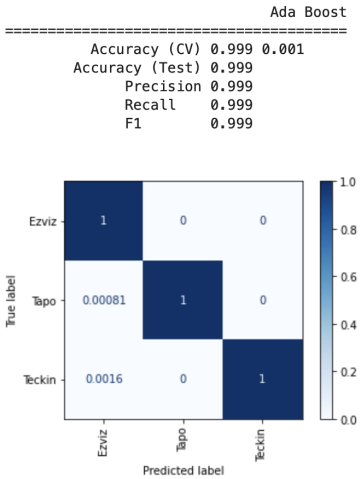

Figure 6.18: Camera classification - Monitor mode - Adaboost classifier

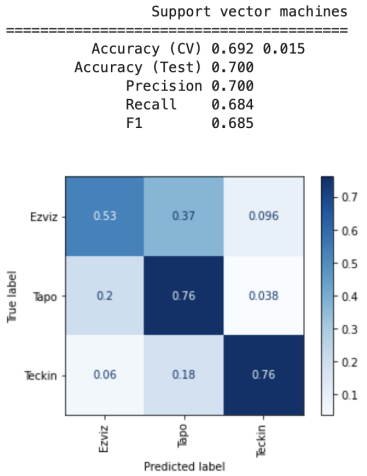

Figure 6.19: Camera classification - Monitor mode - Support Vector Machine classifier

6.8 Comment on camera classification results

Looking at the confusion matrices and at the corresponding performance figures of the two previous sections, we can notice really encouraging results: apart from K-NN and Naive-Bayes, all the other classifiers present an accuracy really close to 1 both in non monitor and monitor mode. As a consequence, we can say that the shape of the traffic sent and received by different cameras is so characteristic that it's almost always possible to understand which camera is transmitting using a proper classifier and the chose features; moreover, this can be done independently on the position of the observer, internal or external, with respect to the system.

6.9 Activity classification for single cameras

In the following sections, I will provide the results obtained by performing activity classification on the single cameras (so, for each camera the classifiers will be trained with the data collected only from the camera under analysis, no data from other cameras) by showing the confusion matrices and the related performances of the different classifiers. In particular:

- Sections 6.9.1 and 6.9.2 describe the results for the Teckin camera respectively in non monitor and monitor mode.
- Sections 6.9.3 and 6.9.4 describe the results for the Tapo camera respectively in non monitor and monitor mode.
- Sections 6.9.5 and 6.9.6 describe the results for the Ezviz camera respectively in non monitor and monitor mode.

6.9.1 Teckin camera - Non monitor mode

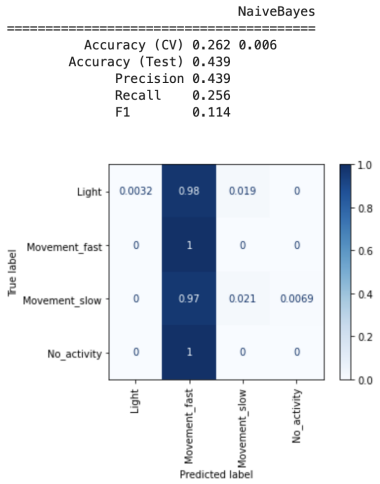

Figure 6.20: Teckin - Activity classification - Non monitor mode - Naive Bayes classifier

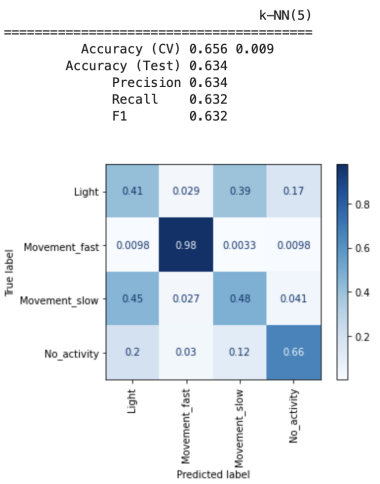

Figure 6.21: Teckin - Activity classification - Non monitor mode - K-NN classifier

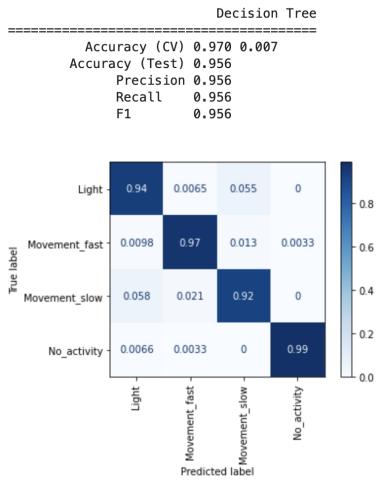

Figure 6.22: Teckin - Activity classification - Non monitor mode - Decision Tree classifier

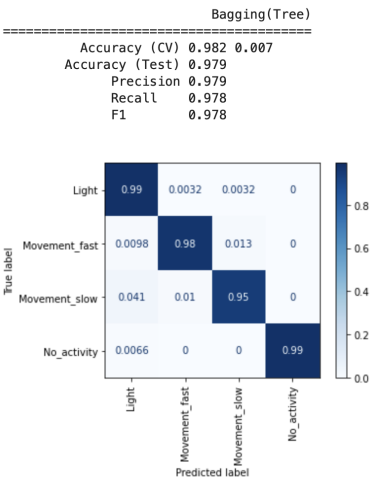

Figure 6.23: Teckin - Activity classification - Non monitor mode - Bagging (tree) classifier

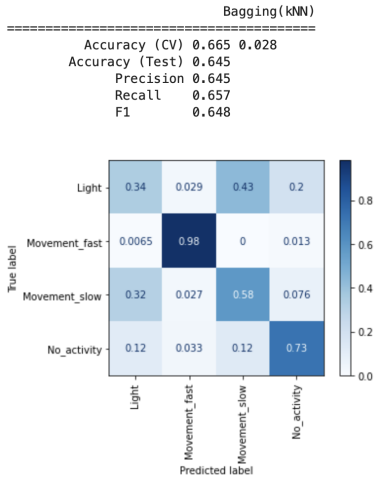

Figure 6.24: Teckin - Activity classification - Non monitor mode - Bagging (KNN) classifier

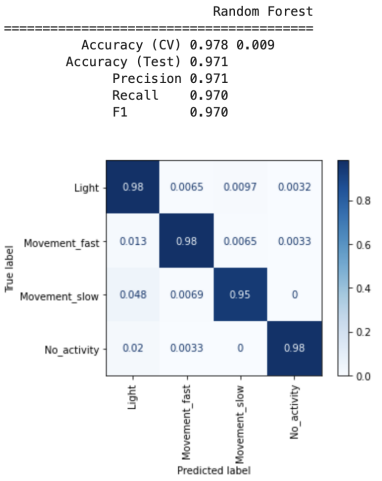

Figure 6.25: Teckin - Activity classification - Non monitor mode - Random Forest classifier

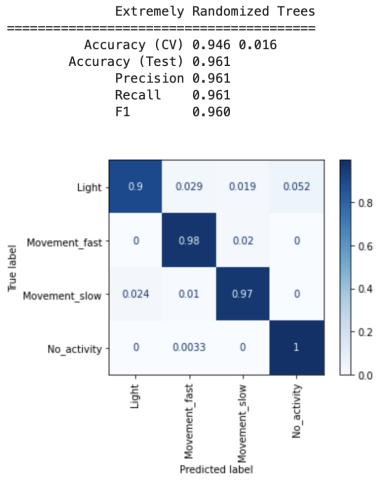

Figure 6.26: Teckin - Activity classification - Non monitor mode - Extremely Randomized Trees classifier

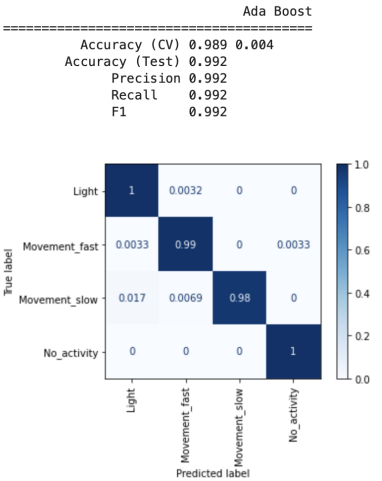

Figure 6.27: Teckin - Activity classification - Non monitor mode - Adaboost classifier

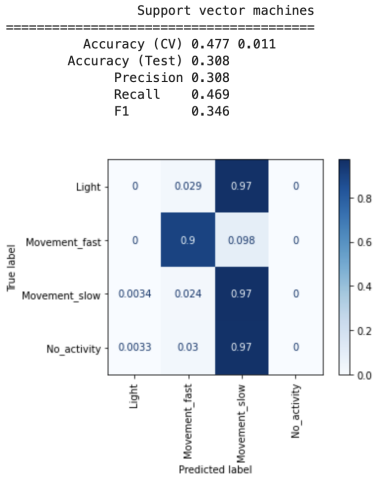

Figure 6.28: Teckin - Activity classification - Non monitor mode - Support Vector Machine classifier

6.9.2 Teckin camera - Monitor mode

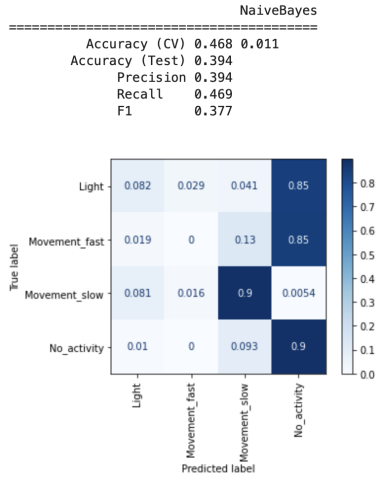

Figure 6.29: Teckin - Activity classification - Monitor mode - Naive Bayes classifier

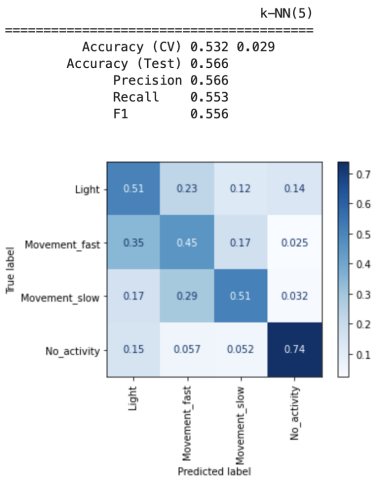

Figure 6.30: Teckin - Activity classification - Monitor mode - K-NN classifier

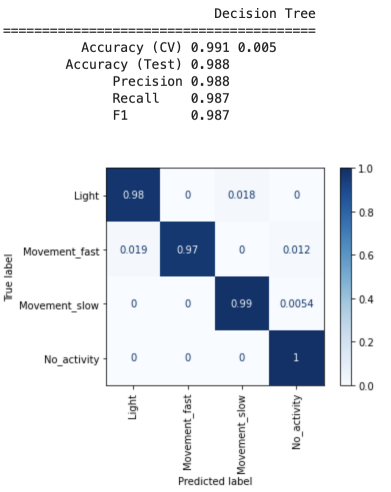

Figure 6.31: Teckin - Activity classification - Monitor mode - Decision Tree classifier

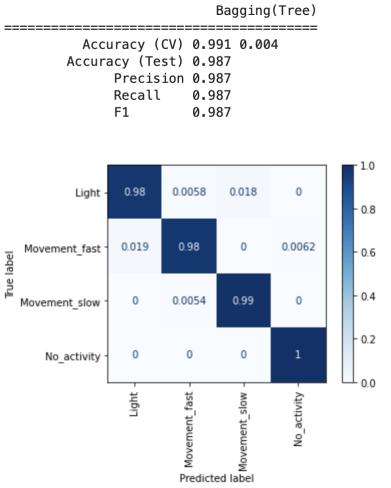

Figure 6.32: Teckin - Activity classification - Monitor mode - Bagging (tree) classifier

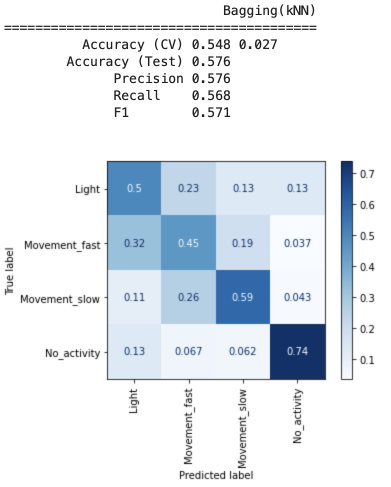

Figure 6.33: Teckin - Activity classification - Monitor mode - Bagging (KNN) classifier

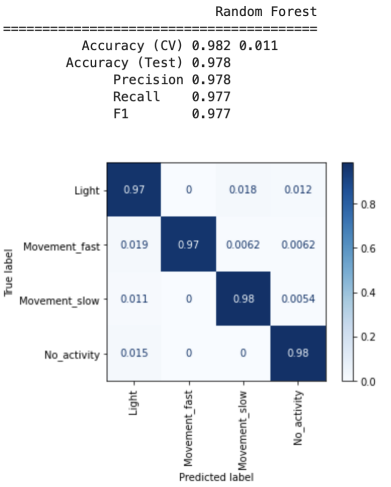

Figure 6.34: Teckin - Activity classification - Monitor mode - Random Forest classifier

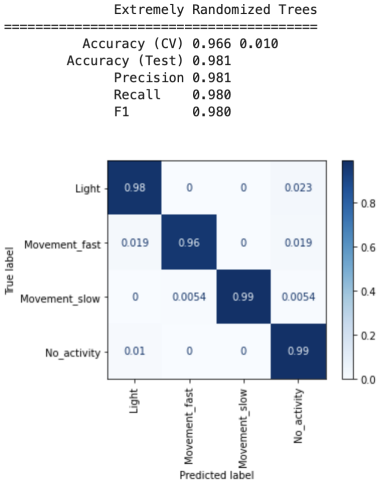

Figure 6.35: Teckin - Activity classification - Monitor mode - Extremely Randomized Trees classifier

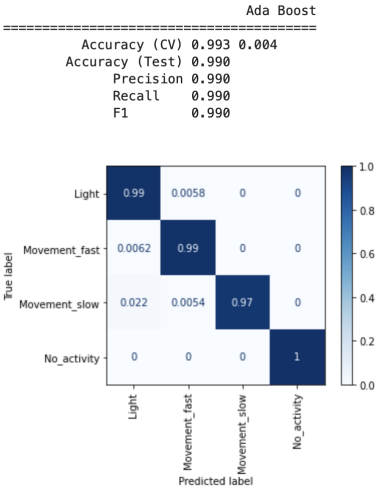

Figure 6.36: Teckin - Activity classification - Monitor mode - Adaboost classifier

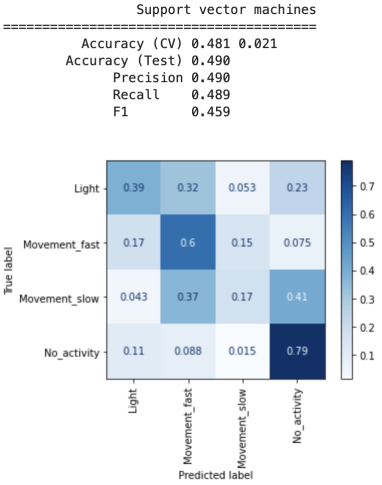

Figure 6.37: Teckin - Activity classification - Monitor mode - Support Vector Machine classifier

6.9.3 Tapo camera - Non monitor mode

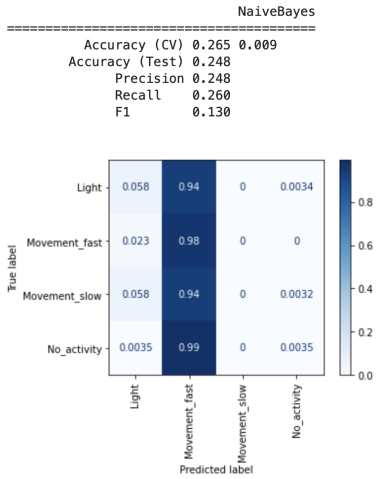

Figure 6.38: Tapo - Activity classification - Non monitor mode - Naive Bayes classifier

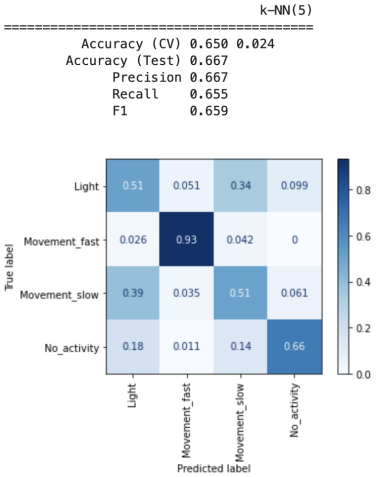

Figure 6.39: Tapo - Activity classification - Non monitor mode - K-NN classifier

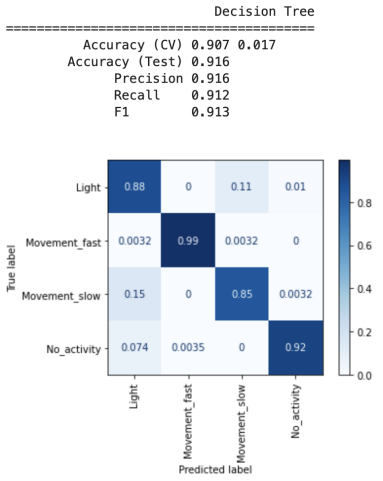

Figure 6.40: Tapo - Activity classification - Non monitor mode - Decision Tree classifier

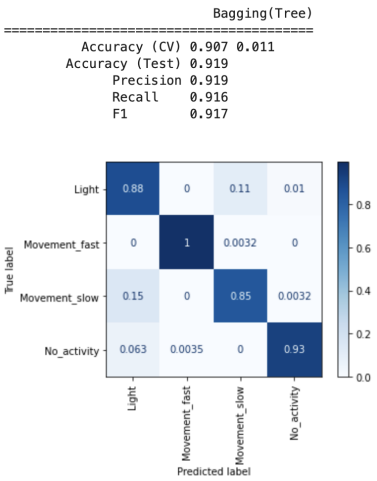

Figure 6.41: Tapo - Activity classification - Non monitor mode - Bagging (tree) classifier

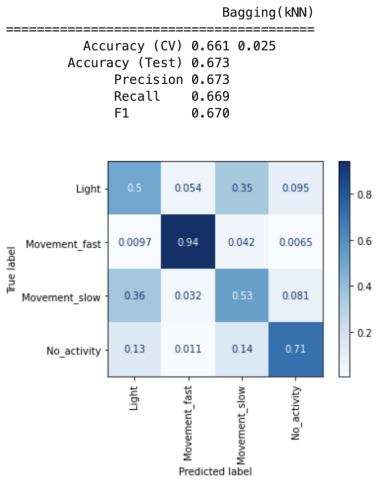

Figure 6.42: Tapo - Activity classification - Non monitor mode - Bagging (KNN) classifier

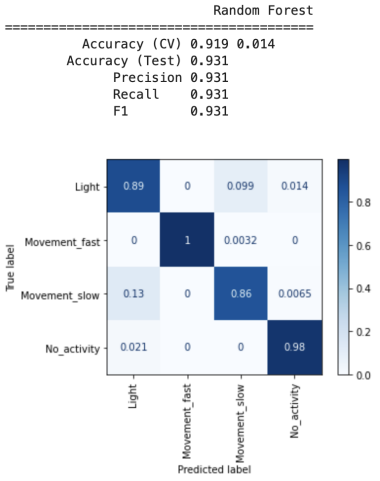

Figure 6.43: Tapo - Activity classification - Non monitor mode - Random Forest classifier

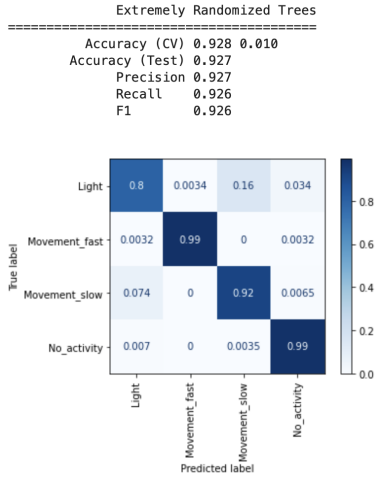

Figure 6.44: Tapo - Activity classification - Non monitor mode - Extremely Randomized Trees classifier

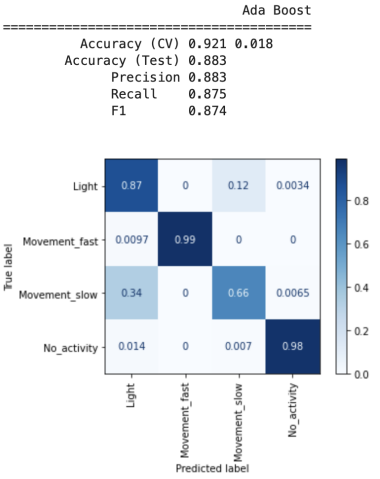

Figure 6.45: Tapo - Activity classification - Non monitor mode - Adaboost classifier

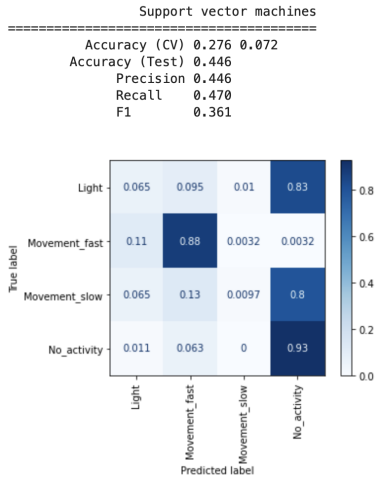

Figure 6.46: Tapo - Activity classification - Non monitor mode - Support Vector Machine classifier

6.9.4 Tapo camera - Monitor mode

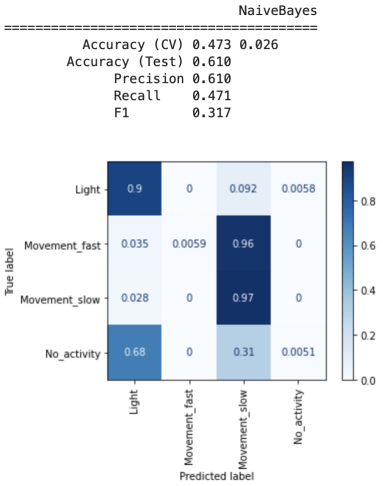

Figure 6.47: Tapo - Activity classification - Monitor mode - Naive Bayes classifier

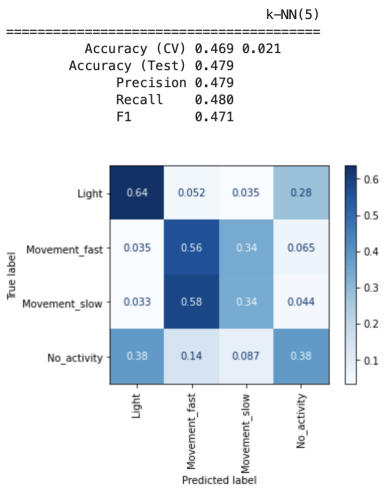

Figure 6.48: Tapo - Activity classification - Monitor mode - K-NN classifier

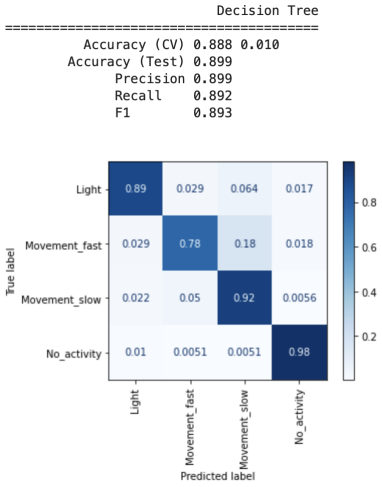

Figure 6.49: Tapo - Activity classification - Monitor mode - Decision Tree classifier

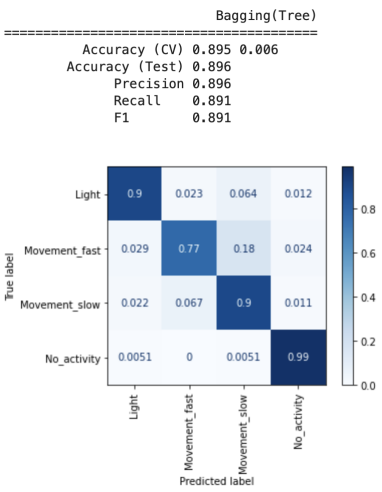

Figure 6.50: Tapo - Activity classification - Monitor mode - Bagging (tree) classifier

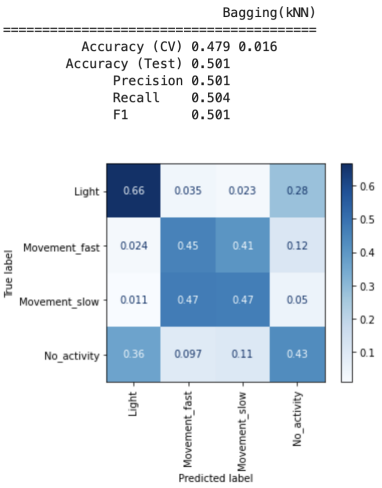

Figure 6.51: Tapo - Activity classification - Monitor mode - Bagging (KNN) classifier

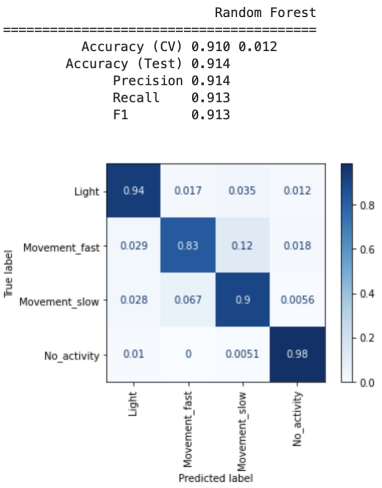

Figure 6.52: Tapo - Activity classification - Monitor mode - Random Forest classifier

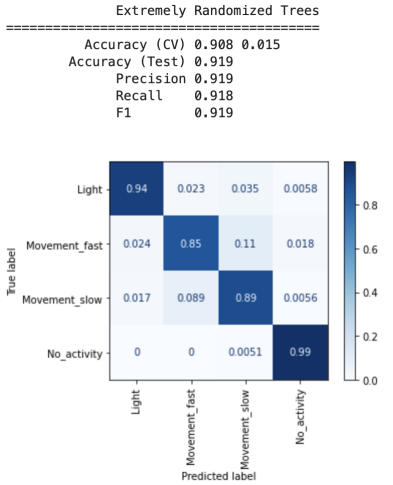

Figure 6.53: Tapo - Activity classification - Monitor mode - Extremely Randomized Trees classifier

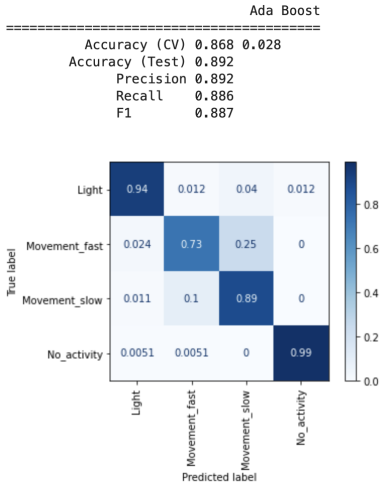

Figure 6.54: Tapo - Activity classification - Monitor mode - Adaboost classifier

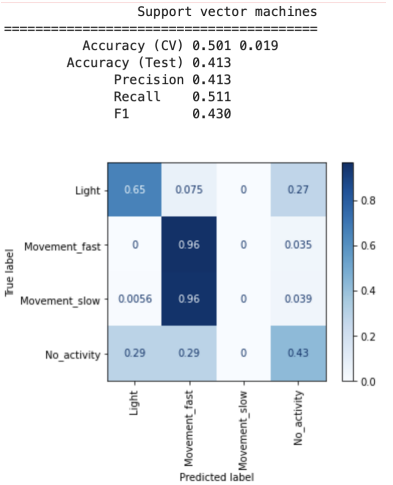

Figure 6.55: Tapo - Activity classification - Monitor mode - Support Vector Machine classifier

6.9.5 Ezviz camera - Non monitor mode

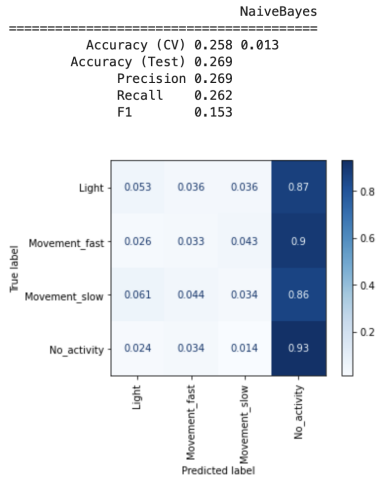

Figure 6.56: Ezviz - Activity classification - Non monitor mode - Naive Bayes classifier

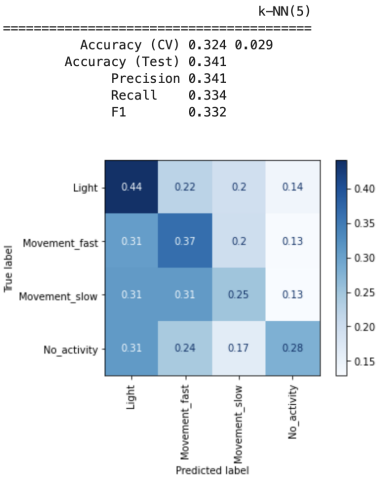

Figure 6.57: Ezviz - Activity classification - Non monitor mode - K-NN classifier

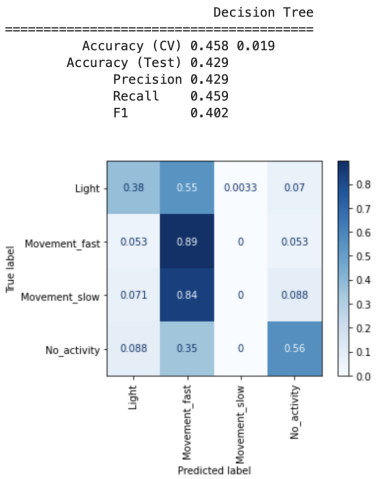

Figure 6.58: Ezviz - Activity classification - Non monitor mode - Decision Tree classifier

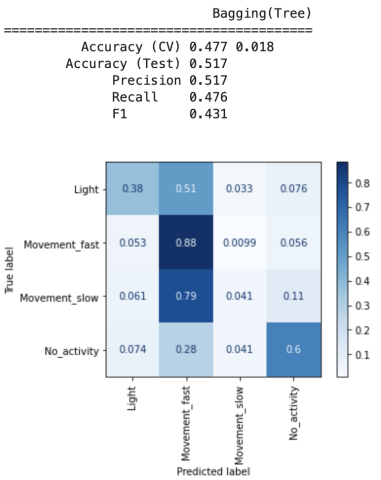

Figure 6.59: Ezviz - Activity classification - Non monitor mode - Bagging (tree) classifier

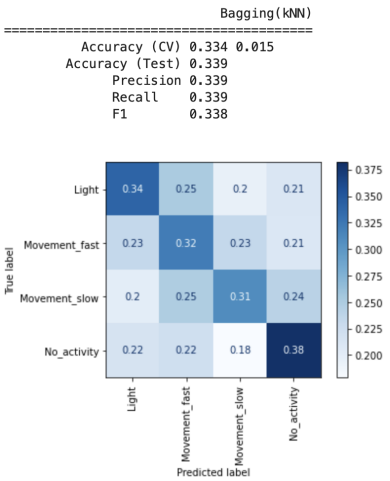

Figure 6.60: Ezviz - Activity classification - Non monitor mode - Bagging (KNN) classifier

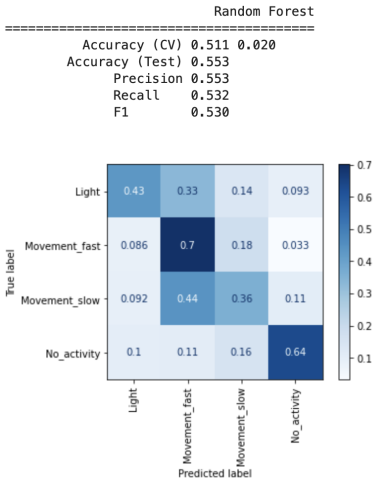

Figure 6.61: Ezviz - Activity classification - Non monitor mode - Random Forest classifier

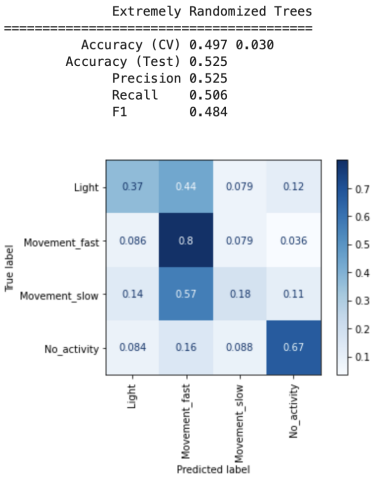

Figure 6.62: Ezviz - Activity classification - Non monitor mode - Extremely Randomized Trees classifier

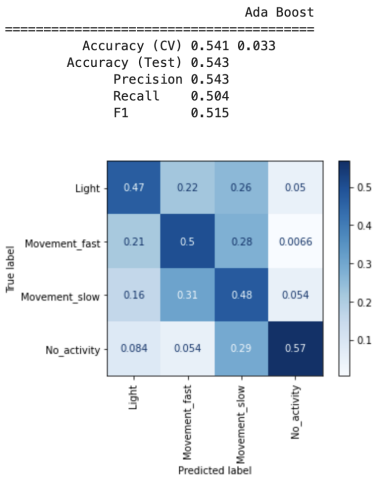

Figure 6.63: Ezviz - Activity classification - Non monitor mode - Adaboost classifier

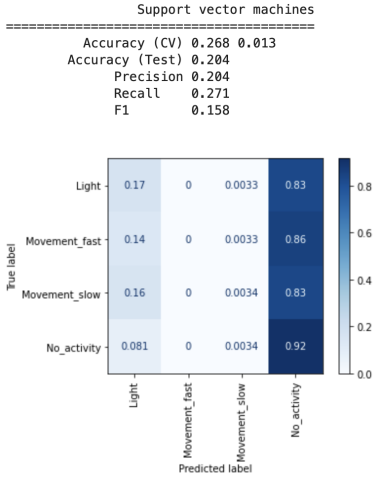

Figure 6.64: Ezviz - Activity classification - Non monitor mode - Support Vector Machine classifier

6.9.6 Ezviz camera - Monitor mode

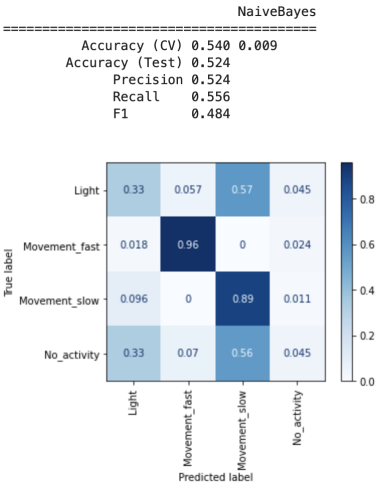

Figure 6.65: Ezviz - Activity classification - Monitor mode - Naive Bayes classifier

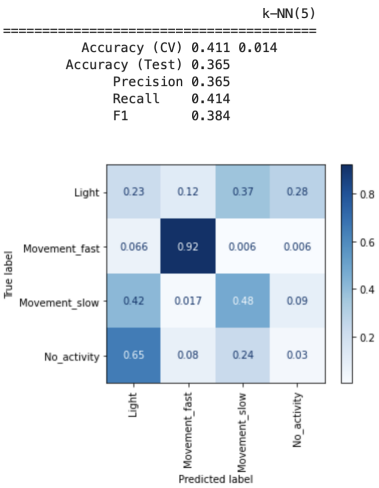

Figure 6.66: Ezviz - Activity classification - Monitor mode - K-NN classifier

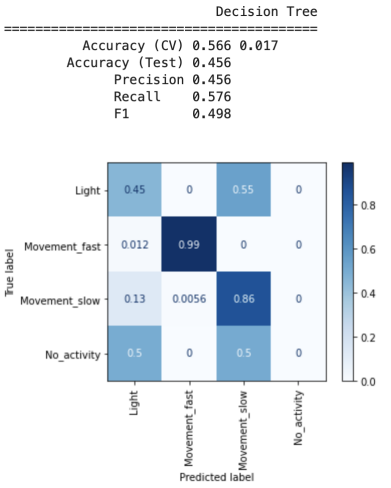

Figure 6.67: Ezviz - Activity classification - Monitor mode - Decision Tree classifier

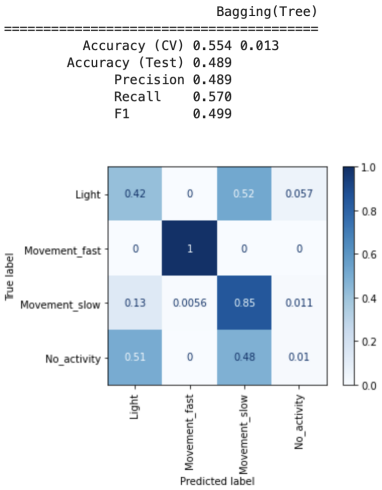

Figure 6.68: Ezviz - Activity classification - Monitor mode - Bagging (tree) classifier

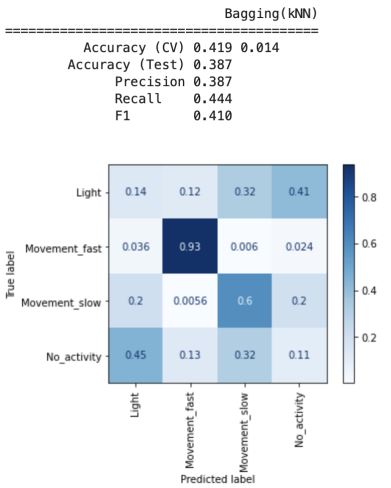

Figure 6.69: Ezviz - Activity classification - Monitor mode - Bagging (KNN) classifier

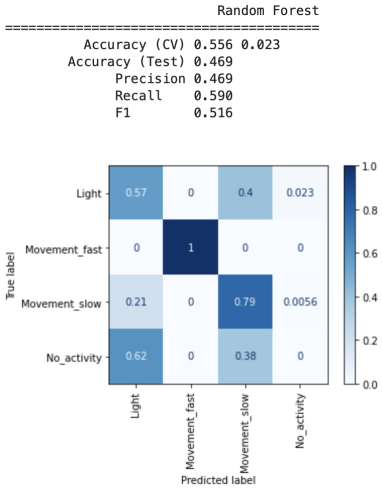

Figure 6.70: Ezviz - Activity classification - Monitor mode - Random Forest classifier

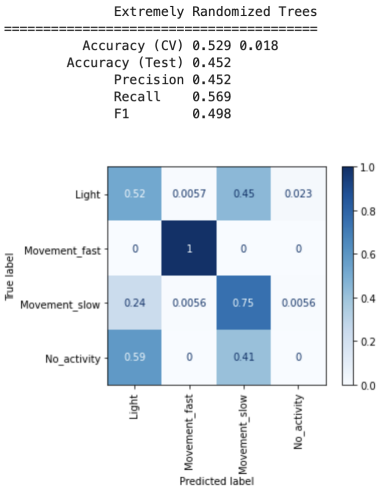

Figure 6.71: Ezviz - Activity classification - Monitor mode - Extremely Randomized Trees classifier

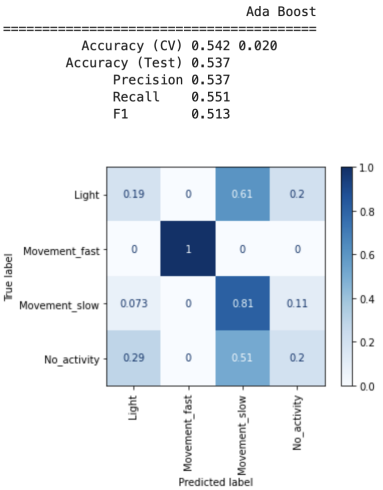

Figure 6.72: Ezviz - Activity classification - Monitor mode - Adaboost classifier

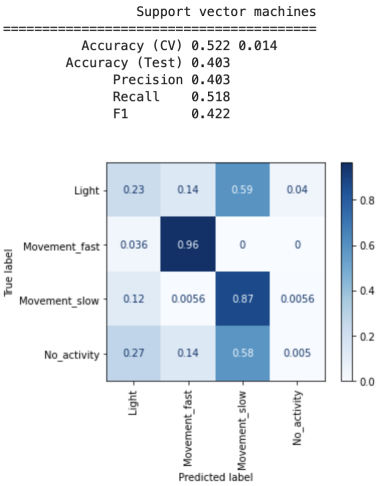

Figure 6.73: Ezviz - Activity classification - Monitor mode - Support Vector Machine classifier

6.10 General comments on activity classification

As it can be noticed by looking at the confusion matrices in the previous sections, the quality of the performance figures is strictly dependent on the classifiers that it's used. In general, the ones with the highest performance are Decision Trees, Random Forests and Adaboost, while the worst are the Naive-Bayes and the K-NN classifiers.

This suggests that, for practical applications, the best choice should be using one of the best three. Actually, in order to avoid the problem of overfitting that may arise using Decision trees (that are in generale very subject to overfitting), it's better to use one between Random Forest and Adaboost.

Moreover, the results for the Teckin and Tapo camera are much better than the ones for the Ezviz camera. Actually, this is something we could expect. If we go back to the images in Section 3.5, we can see that the traffic size pattern for all the activities for the Ezviz camera is less defined that the one for the other two cameras, meaning that there is less difference between the traffic when there is no activity and when something is happening in the FoV of the camera and, given that something is indeed happening, the graphs are very similar. This means that there is less variance in the features of the traffic of the Ezviz camera than for the Teckin and Tapo.
Finally, as expected, the performances of the classifications in monitor and non monitor mode are almost the same. This is good, since it means that, forensically speaking, is in principle possible to retrieve the same information (se, the same evidences) both from the inside and from the outside of the system.

6.11 Activity classification for all the cameras

A question that one may ask is: what if we don't have a classifier trained for a specific camera but we need to make a classification of some activity in the FoV of it? Is it still possible to use the knowledge that we have to perform activity classification?
In order to try answering this question, I have trained and tested different classifiers using as dataset one resulting from the merge of all the datasets of all the cameras. Sections 6.11.1 and 6.11.2 shows the results respectively in non monitor and monitor mode.
Commenting the results that follows, we can notice that the performance are not bad, but not as good as the ones obtained by using for each camera its own activity classifier. Indeed, this degradation of the performance is expected since we have already noticed many times that different cameras produce different traffic patterns, so each one should have its proper classifiers.

6.11.1 Non monitor mode

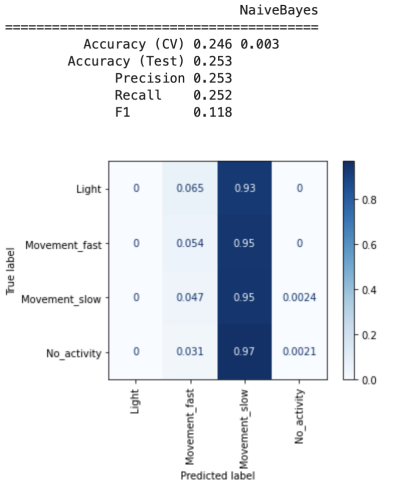

Figure 6.74: Activity classification - Non monitor mode - Naive Bayes classifier

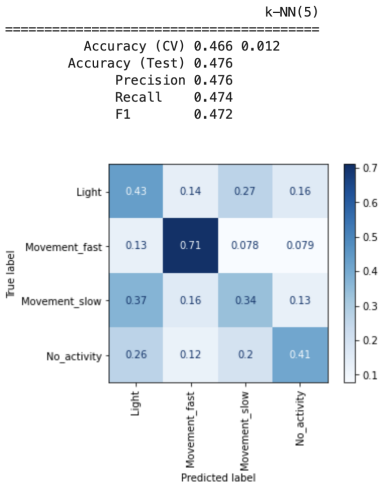

Figure 6.75: Activity classification - Non monitor mode - K-NN classifier

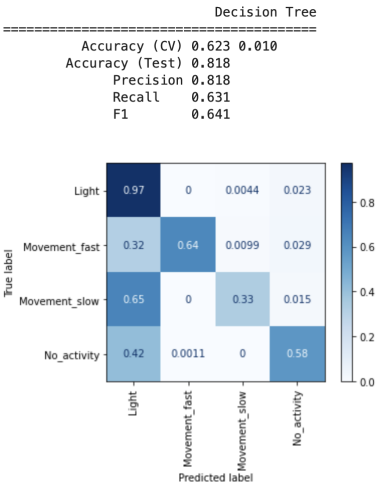

Figure 6.76: Activity classification - Non monitor mode - Decision Tree classifier

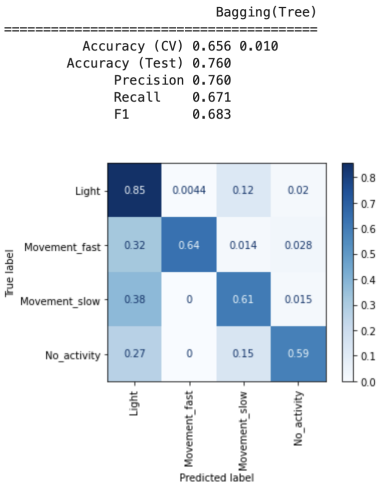

Figure 6.77: Activity classification - Non monitor mode - Bagging (tree) classifier

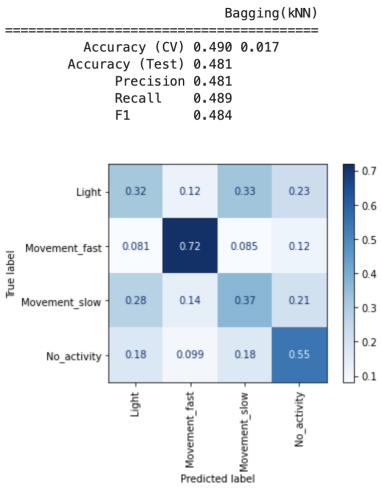

Figure 6.78: Activity classification - Non monitor mode - Bagging (KNN) classifier

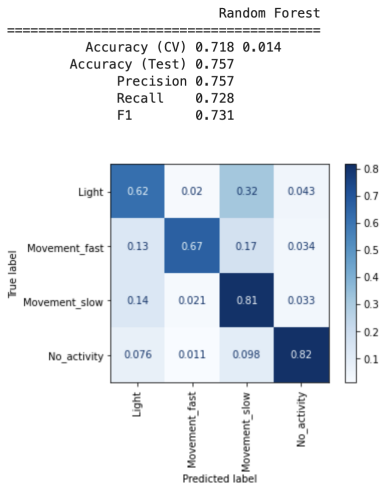

Figure 6.79: Activity classification - Non monitor mode - Random Forest classifier

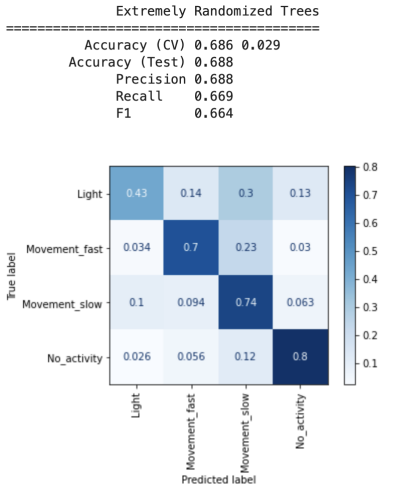

Figure 6.80: Activity classification - Non monitor mode - Extremely Randomized Trees classifier

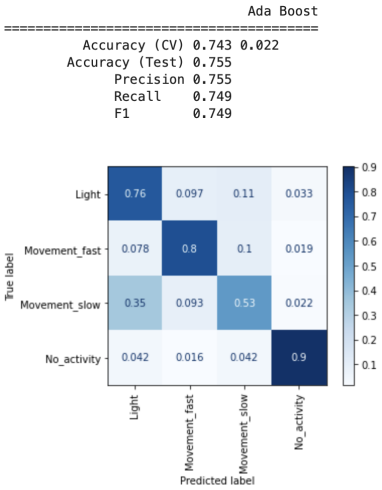

Figure 6.81: Activity classification - Non monitor mode - Adaboost classifier

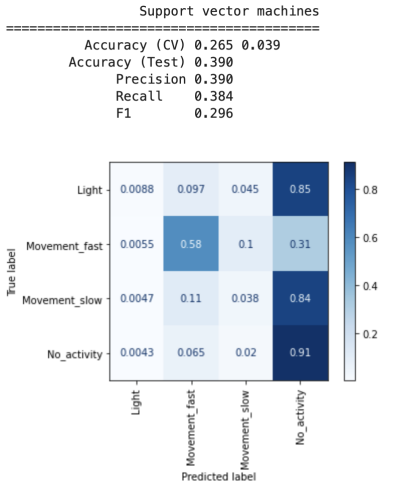

Figure 6.82: Activity classification - Non monitor mode - Support Vector Machine classifier

6.11.2 Monitor mode

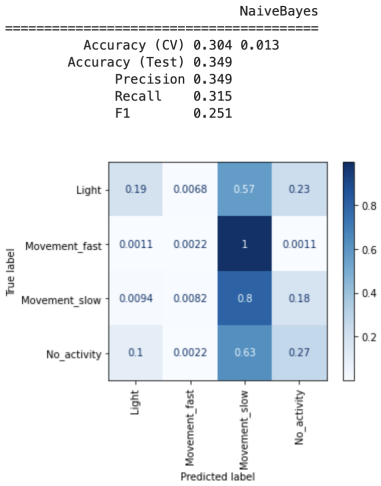

Figure 6.83: Activity classification - Monitor mode - Naive Bayes classifier

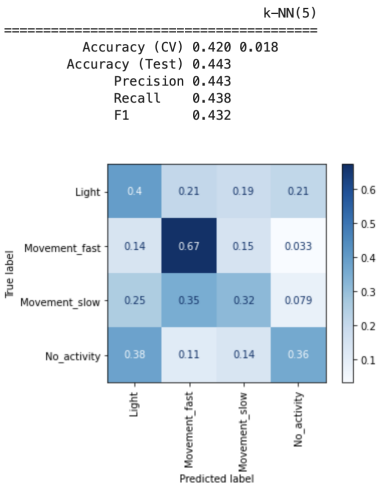

Figure 6.84: Activity classification - Monitor mode - K-NN classifier

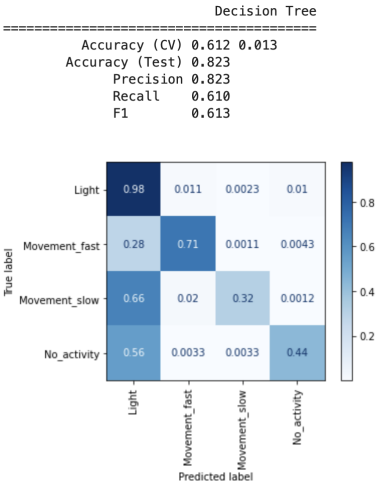

Figure 6.85: Activity classification - Monitor mode - Decision Tree classifier

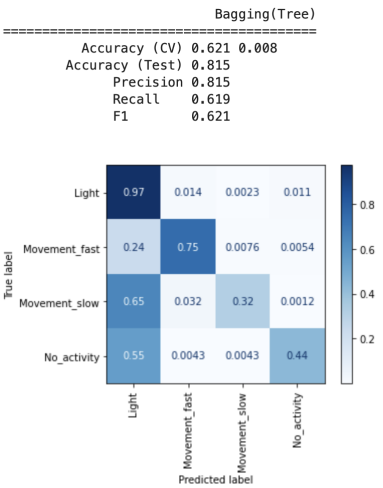

Figure 6.86: Activity classification - Monitor mode - Bagging (tree) classifier

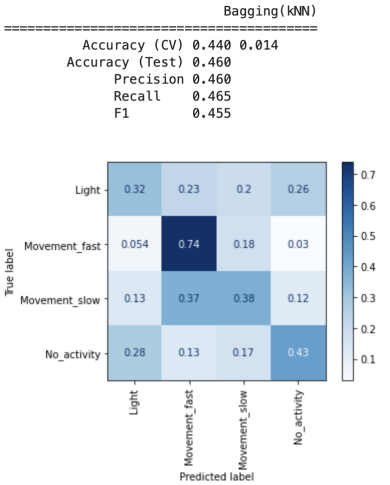

Figure 6.87: Activity classification - Monitor mode - Bagging (KNN) classifier

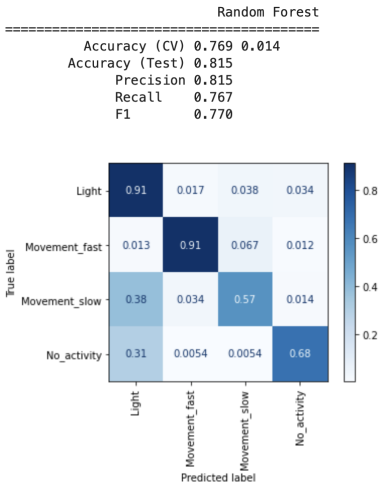

Figure 6.88: Activity classification - Monitor mode - Random Forest classifier

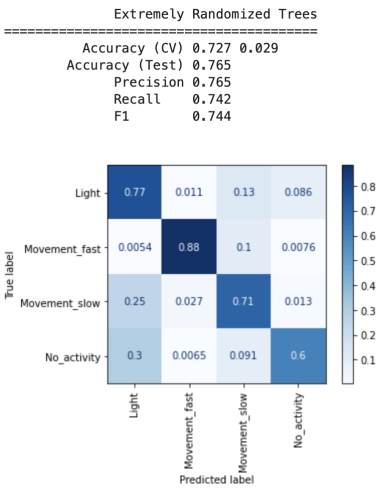

Figure 6.89: Activity classification - Monitor mode - Extremely Randomized Trees classifier

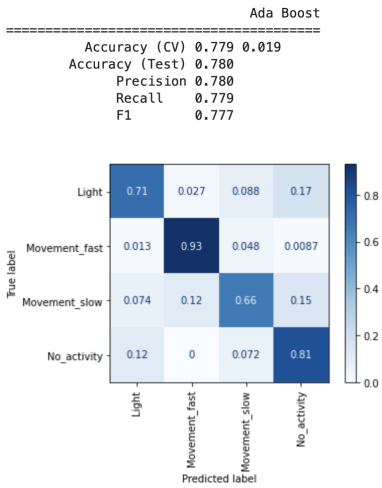

Figure 6.90: Activity classification - Monitor mode - Adaboost classifier

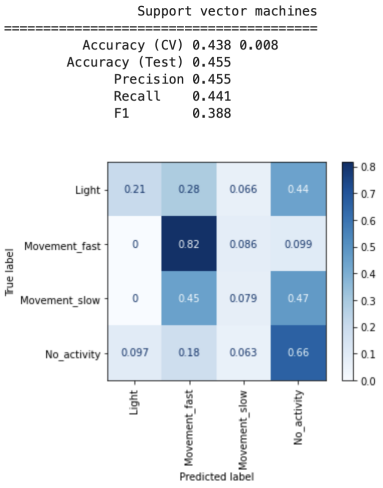

Figure 6.91: Activity classification - Monitor mode - Support Vector Machine classifier

6.12 Turning on input and output audio: what changes?

All the cameras that have been used are provided with built in microphone and speaker so that it's possible to ear the noises in the room where a camera is placed and to ear from the camera someone that is talking to the application of the camera on their phone.
Of course, when the two-way audio feature is enabled on a camera, in addition to the video stream there is the audio data stream sent and/or received by the camera. Therefore, the question arises: can we use the classifiers trained using only video data to classify an activity happening in the FoV of a camera when the audio is active?
To answer this question, we need to do some additional things:

1. First, for each camera, we need to save the classifier(s) we want to use for classifying the data obtained with the audio enabled (in this case, I have made the test using the Random Forest classifier).

2. Then, we need to collected new data by keeping the audio feature enabled and performing the same activities but with some audio too. This has been done by playing a song in background in the room where the camera is placed (note that this way only the uplink traffic will be subject to some variations).

3. At this point, we need to process the new capture files so to extract the features from them, apply to them the same preprocessing described in Section 6.2 and proceed with the classification.

Sections 6.12.1 and 6.12.2 we can see, respectively in non monitor and monitor mode, how the accuracy of the classificarion gets worse when the audio is enabled for all the cameras. This degradation is due to the additional audio traffic (which cannot be distinguished by the video traffic from the outside) causes an increase in the transmission rate that tends to confuse the classifiers.

6.12.1 Non monitor mode

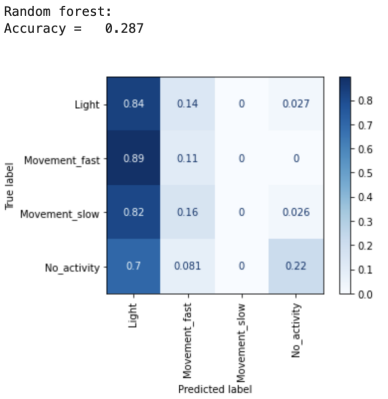

Figure 6.92: Teckin - Activity classification with audio - Non monitor mode

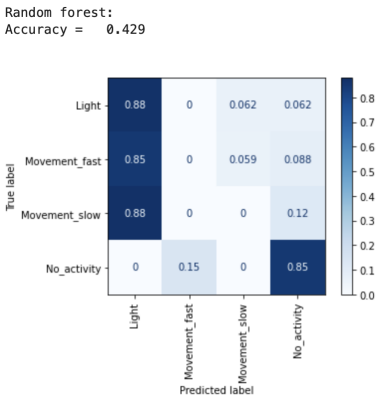

Figure 6.93: Tapo - Activity classification with audio - Non monitor mode

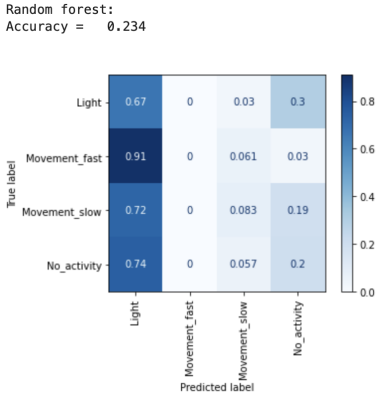

Figure 6.94: Ezviz - Activity classification with audio - Non monitor mode

6.12.2 Monitor mode

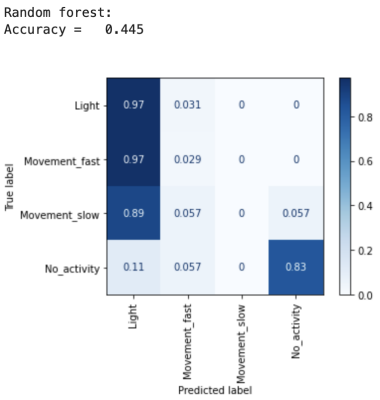

Figure 6.95: Teckin - Activity classification with audio - Monitor mode

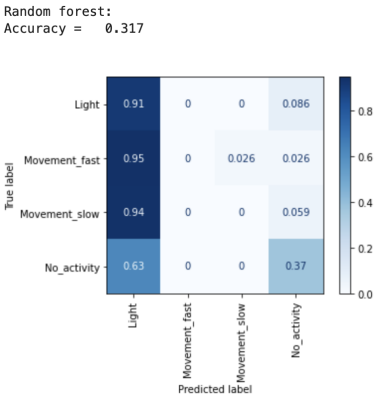

Figure 6.96: Tapo - Activity classification with audio - Monitor mode

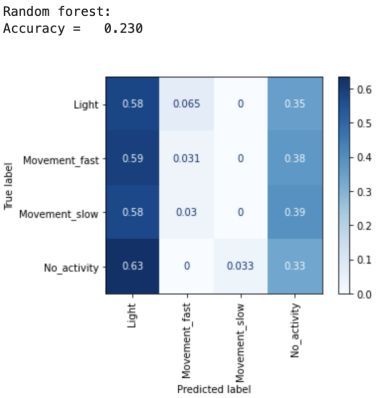

Figure 6.97: Ezviz - Activity classification with audio - Monitor mode

Chapter 7

Practical usage of classification: a live analysis approach

At this point, we have understood that, in theory, we can indeed understand with a certain level of accuracy (which depends on the classifier we are using) two things:

1. Given that one of the cameras we have been able to collect data from is transmitting, which model between the known ones is indeed transmitting.
2. Once understood which camera is sending/receiving traffic, which activity (again, between the known ones) is being performed in the FoV of the camera.

Now, the matter is: given the accuracy with which we can perform these two classifications, can we implement a mechanism that does this classification automatically? Of course, the answer yes, or at least we can try.

7.1 Assumptions and implications

Before going on, we need to clarify some points.
Remember that we have trained the classifiers in a supervised way, that is, we have fed them with some data and we have them let understand how to use the information contained in the data to distinguish between three different models of cameras and four different activities for each of the cameras. This implies two things:

- Our live analysis will work only when the camera that is transmitting/receiving is one among the known ones, where this means that we are supposing we know the possible MAC addresses that may be transmitting/receiving traffic. This is quite a strong assumption, since it could be rare for the forensic professional to have such a specific information as the MAC address of a device. In fact, in order to have this knowledge , the forensic professional should be the proprietary of the network: if they are external (so being able to retrieve information up to the MAC layer), it's unlikely they would know the address of the transmitting device.
- We also suppose that there is a single cam transmitting, which may in principle not be true especially in an urban environment where many places might be using an IP camera.

- Since we have trained our models to distinguish between four activities, the maximum granularity with which we can reconstruct a user's activity is the one of these four activities.

These implications are quite strong, in the sense that they may limit the possibilities of a camera and activity classifier in the real world. Anyway, for the purposes of this research project, a controlled environment like the one just described in more than sufficient. Moreover, it's not difficult (at least in principle) to extend this work so to prepare it for the open world.

7.2 Saving the models

Of course, in order to perform the classification outside of the train and test phase, we need to save the models that we want to use.
For having better and more trained models, what I have done is training different models using all the data that I have.
Once trained, the models have been saved using the *Pickle* python library.
The models I have saved for performing the live analysis are the ones that have shown the best performances in the test phase, so Decision Trees, Random Forests and Adaboost. For the Decision Trees, in order to proactively try to avoid overfitting problems, I have tried to use trees with different depths, in particular I have used trees with depth equal to 3, 4, 5 and *None*.

7.3 Structure of the live analysis

The program performing the live analysis of the traffic of a camera can be summarized as follows:

1. Importing the libraries needed, the most important ones are *sklearn*, *pikle* and *pandas*.

2. Declaration of the constants, so of all the known MAC addresses and of the time window for feature extraction.

3. Declaration of the variables needed for performing the feature extraction procedure. Such variables are indeed of the same type described in Section 5.5, with the difference that instead having one variable for each camera we have a single variable for the kind (features, inter arrival time, traffic size).

4. Importation of the classifiers that we need, so the one for the camera classification and the three ones for the activity classification (one for each camera).

5. Data collection procedure: the user is asked for how long they want to sniff the Wi-Fi channel in seconds (note: the program doesn't allow to select also the Wi-Fi channel on which to sniff, so the user must be sure they are already synchronized to the one they're interested in). The channel is then observed for a time equal to the one specified by the user, and this is done by making a system call to *tshark*, which is another tool that can be used to make captures. The system call is like the following:

'sudo tshark -Ini ' + interface + ' -a duration:' + sniff_time + ' -w ' + cap_file

Here, *sudo* is used to tun the command with root privileges, *-Ini + interface* is used to specify the interface on which to sniff, *-a duration: + sniff_time* fixes the the stopping condition for the command after *sniff_time* seconds (the value is the one specified by the user) and *-w + cap_file* states that the file will be saved in the position specified by *cap_file*.
Note that the command *tshark* produces a capture in monitor mode; if we want work in non monitor mode, we just have to use the command *tcpdump* as described in Section 5.2

6. Once the capture is over, we proceed with the feature extraction procedure, performed the exact same way described in Section 5.5. At the end, the dataset is saved and preprocessed as described in Section 6.2.

7. Now, there is the classification phase: each line of the dataset is analyzed and, if there are no meaningless values (that is, values equal to -1), the row is passed to the camera classifier and then to the activity classifier of the camera that is the output of the camera classification phase.

The output of the classification is written in a text (*.txt*) file according to this format:

Time: [Time window] –> Camera: [Camera] –> Activity: [Activity]

The *Time window* value within square brackets indicates the time in seconds from the beginning of the capture to which the prediction is referred; the following values of the *Camera* and *Activity* between square brackets are the ones predicted respectively by the camera and the activity classifiers.

7.4 Results

The program for the live analysis has been tested only in monitor mode. This because, in order to try it in non monitor mode, it would be necessary to run it on the Raspberry and not on the laptop.
The results obtained in monitor mode were not as good as we could expect looking at the theoretical performance of the classification in test phase.
While the camera classification was almost never wrong (for this, it's been always used the Decision Tree classifier), the activity identification created some problems and sometimes the classification output was not right. This may be due to some overfitting issues in the Decision Trees used.
If we want to try to forecast what would happen in non monitor mode, since it must work at least as well as in monitor mode, we can say that the camera classification would probably be correct, but we cannot make accurate prediction on the activity identification.

7.5 Removing assumptions: hypothesis on generalization

As described earlier, the live analysis I have performed comes with as assumption the knowledge of the MAC addresses of the cameras that may be transmitting, and it's been already commented how strong this assumption is.
Actually, it is in principle possible to remove this assumption. In fact, it's known that, given a MAC address, the first three couples of figures of the address itself

are related to the vendor of the device. This is an information that can be used: for instance, we can produce a file containing all addresses of all the vendors of IP cameras (or at least of the ones we have the classifiers ready for performing activity identification) and check if the source/destination MAC address of some packets belongs to the file.
Of course, since we are in a supervised learning framework, in order to perform both camera and activity classification, we need to train the classifiers, which implies the collection of a lot of data.
Regarding the assumption on the single camera that is transmitting, this is not difficult at all to remove. In case a MAC address of a vendor of IP cameras is transmitting/receiving, just check if it was already revealed or not: if not, it's sufficient to declare a new variable for extracting its features.
Finally, if we want to distinguish among more activities, just collect data and train the classifiers.

Chapter 8

Conclusions

In this work, we have deeply explored the world of IP smart cloud cameras.
We have started by understanding how they work, how they encode data for the transmission and that these have some properties that we could use to obtain some evidences for forensics purposes.
Then, we have successfully tried to actually retrieve some knowledge using the properties of differential video encoding and found an effective strategy that can be used to find such knowledge, passing from building a system for collecting data from scratch, moving to the actual collection of data and finally to the choice of the statistics of the input/output traffic in which the information we were looking for may be hidden.
Finally, we have seen which ML algorithm may be the best in order to distinguish among some cameras and activities and we have designed and implemented an algorithm for performing camera and activity classification live; also, we have discussed how the implemented algorithm can be extended to deal with real world scenarios.

At this point, there still is one question left: is it always possible to perform the classifications we have performed? The answer is no: there are cameras on the market whose transmission is specifically designed to avoid the possibility of inferring the information we may be forensically speaking interested in.
More specifically, these cameras are designed to transmit packets all of the same dimension and at time intervals that are kept as more constant as possible. As a consequence, the features related to the traffic size cannot be directly used (since the variance of those features would be null, making the features useless) and the ones related to the inter-arrival time and, consequently, to the rate, would have a very low variance, making the classification very hard if not impossible with high accuracy.
An example of camera that implement these functionalities is the *Amazon Blink* cameras. In Figure 8.1, we can indeed see that in the traffic size there are no peaks identifying the presence of I frames nor intervals in which the traffic size is larger or smaller:it's just constant, so not allowing us to retrieve any information.

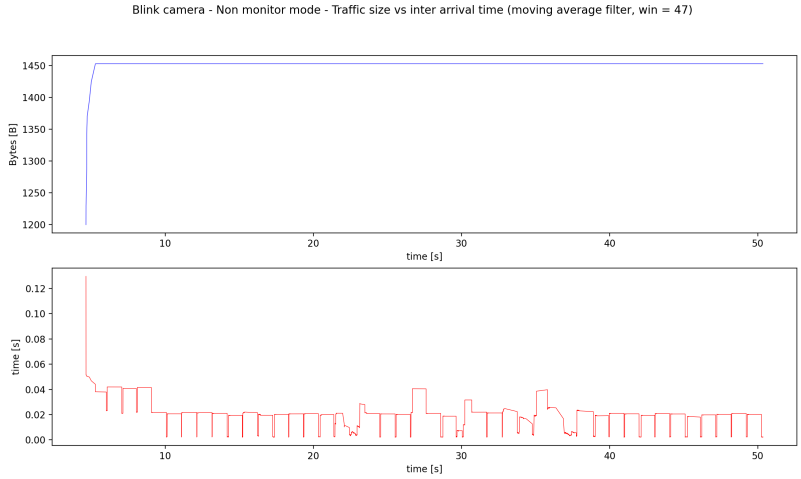

Figure 8.1: Blink camere traffic size and inter-arrival time

Anyway, the number of commercial cameras that performs a kind of traffic shaping as the Blink camera is low on the market, and the ones that actually do this are much more expensive than the ones that don't (the price of the Blink camera system is more that the double than any of the other camera we have used.

8.1 Future works

If we look at the results of this work, we may say that it might be clever to develop a more complex system that can be used to perform camera and activity classification. Here, more complex means that it would be interesting to:

1. Expand the work to more cameras.

2. Expand the work to more activities.

3. Try to use other approaches for classification, such as deep learning algorithms or artificial intelligences.

Expanding this work to more cameras and/or activities requires to collect new data, so a lot of time. Vice versa, the implementation of other approaches for performing this type of classification requires the development of new algorithms.

Anyway, there could be a problem: as I have shown, it's not always possible to perform the classifications of this work since some cameras already implement traffic shaping techniques to prevent an observer from inferring the information I have demonstrated that can be obtained with this work. Nowadays, the cameras implementing this features are a few and more expensive than the average, but in the future it may happen that, with the development of the technologies, this technology may become the average one, therefore making impossible to perform camera and activity classification.
So, the final question one has to answer is: is it worth to spend time in collecting data and training classifiers even if in some years this kind of work may become useless?

8.2 Final conclusions

In conclusion, we can state that the development of the IoT forensics discipline in the world of IP smart cloud cameras is worthy (at least for now): even if it's expensive from the point of view of the time needed to collect data for training the classifiers needed, the level of accuracy we have reached is an incentive for spending this time so to allow the forensic professionals having new evidences for supporting their thesis and, possibly, help in a faithful reconstruction of the events.

Chapter 9

Bibliography

Bibliography

[1] Chris Wampler, Selcuk Uluagac, and Raheem Beyah, *Information Leakage in Encrypted IP Video Traffic*, IEEE, 2015

[2] Hong Li1, Yunhua He, Limin Sun, Xiuzhen Cheng, Jiguo Yu4, *Side-Channel Information Leakage of Encrypted Video Stream in Video Surveillance Systems*, IEEE, 2016

[3] Jibao Wang, Zigang Cao, Cuicui Kang, Gang Xiong, *User Behavior Classification in Encrypted Cloud Camera Traffic*, IEEE, 2020

[4] Maria Stoyanova, Yannis Nikoloudakis, Spyridon Panagiotakis, Evangelos Pallis, Evangelos K. Markakis, *A Survey on the Internet of Things (IoT) Forensics: Challenges, Approaches, and Open Issues*, IEEE, 2020

[5] Áine MacDermott, Thar Baker, Qi Shi, *IoT Forensics: Challenges For The IoA Era*, IEEE, 2018

[6] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, Nick Feamster, *Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic*

[7] Ibbad Hafeez, Markku Antikainen, Sasu Tarkoma *Protecting IoT-environments against Traffic Analysis Attacks with Traffic Morphing*, IEEE, 2019