Smart Surveillance System on Raspberry-Pi

Politecnico di Torino - Machine Learning for IoT

Francesco Di Salvo
s282418@studenti.polito.it

Gianluca La Malfa
s290187@studenti.polito.it

Leonardo Maggio
s292938@studenti.polito.it

1 Introduction

Video surveillance is the act of monitoring the activities, behavior and changing activities in a scene for the purpose of managing, directing or identifying security threats in an automated way. The use of the advanced computer technology for data acquisition and analysis of those data using sophisticated vision algorithm to capture the unusual activities or directing, guiding and planning the scene under observation without the involvement of human effort is the ideal goal of video surveillance. With the proper use of cameras, security threats like theft, robbery, terrorism or other criminal activities may be controlled.

Edge computing is the idea of moving computations away from the cloud and instead perform them at the edge of the network. The benefits of edge computing are reduced latency, increased integrity, and less strain on networks. Edge AI is the practice of deploying machine learning algorithms to perform computations on the edge.

The vast majority of modern surveillance solutions involve a camera and motion sensors, and just a few of them use artificial intelligence algorithms. In this context, we decided to build an indoor video surveillance system capable of recognizing the presence of a human intrusion, rather than mere movement. In this way, a photo of the intruder can be taken instantly, eliminating the burden of reviewing the footage.

When a machine is able to understand a monitored scene and warn about unusual responses, very large area can be observed and controlled using large number of video sensors by a single person. Hence, continuous and focused monitoring of a very large area becomes possible at a low cost. To make this surveillance system useful and immediate for the user who adopts it in his home, the warnings of possible intrusions are notified directly to his smartphone in real time.

In addition, our video surveillance system is also equipped with a microphone. As a matter of fact, our system is able to classify sounds and images and give an alarm in case suspicious events are detected. An intrusion can be prevented by hearing someone force the door with a hammer or drill. Moreover, perceiving sounds allows to cover those areas that the camera does not reach: an alarm can be given for the breaking of a glass of a window that is not seen by the camera.

The full code of the project is available on GitHub [1].

1.1 Hardware

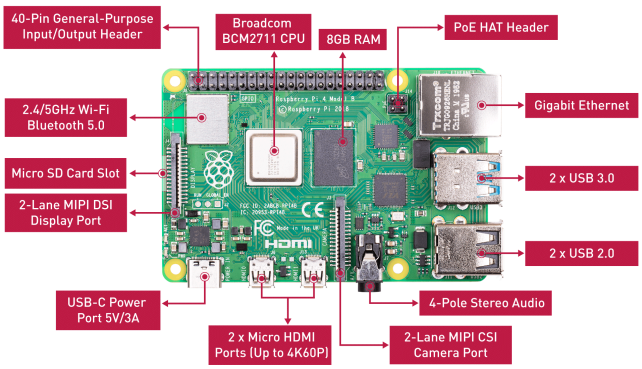

Figure 1: Raspberry Pi 4b Overview.

[1] https://github.com/francescodisalvo05/smart-surveillance-raspberrypi

The heart of the video surveillance system is the Raspberry Pi 4b equipped with 8GB of Ram. It offers ground-breaking increases in processor speed, multimedia performance, memory, and connectivity compared to the prior-generation Raspberry Pi 3 Model B+, while retaining backwards compatibility and similar power consumption. For the end user, Raspberry Pi 4 Model B provides desktop performance comparable to entry-level x86 PC systems.

Its main components are the 64-bit quad-core processor, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0 [12]. Even though we tested our system on the 8GB version, we assume that comparable results can be achieved even with the 4GB version. Several different operative systems can be used on a Raspberry Pi, even dedicated ones for (not-smart) video surveillance systems like MotionEyeOS [11]. However, due to the different goal of the project, we used Raspberry Pi OS (previously called Raspbian) which is the recommended operating system for normal use on a Raspberry Pi.

Moreover, the system needs a camera and a microphone, and any device compatible with Raspberry will be sufficient. In fact, we used an AZDelivery camera (5 MegaPixel) and a Gyvazla USB microphone.

1.2 Devices on the market

For some years now, the IT market has been offering security devices that allow the user to see what is happening at home from a smartphone. However, there are still very few devices that use AI in a strategic manner. Two of the most important ones are:

- *Amazon's Alexa Guard*, a system which provides intelligent alerts for the sounds that the user marked as dangerous during the setup phase [2]. Whenever a suspicious sound is heard, the assistant sends a 10-second audio clip to the user's phone. Then it is up to the user to decide whether to listen to what is happening in the house through a drop-in function or to call an emergency service for help. Smart lights or sirens can be connected and activated as a deterrent in case of danger.

  Alexa Guard is trained to detect smoke alarms and a few suspicious sounds of activity, which are "footsteps", "door closing" and "glass breaking". Alexa comes with Amazon Echo devices starting at $30. However, the Guard service must be activated by the user, who can choose the Free plan which detects only broken glass, or the Plus plan which also recognizes sounds of suspicious activity for $5 per month.

- *Google Nest cam*, an intelligent camera device which performs object recognition tasks on humans, animals and vehicles [7]. It uses a simple but efficient 2 megapixel photomicroscope which can record and provide videos with a quality up to 1080p. It gives the possibility to the user to access to live and recorded videos up to three hours. During its functioning it sends a smart alert whenever an activity is recognized. the google nest camera starts at $90, but a subscription of $5 a month is required to coordinate different devices.

Figure 2: Amazon's Echo dot and Google Nest cam.

2 Communication Paradigm

The devices need to communicate asynchronously, therefore the interactions among the agents are regulated through the MQTT (Message Queue Telemetry Transport) protocol, a publish- subscribe messaging protocol [1]. Each message is published under a certain topic, hierarchically managed through the symbol '/'.

Our application consists of two publishers - one for the camera and one for the microphone - and one subscriber. The former publisher will publish under '/devices/C0001/' and the latter will publish under '/devices/M0001/'. This topic-structure appears to be the most suitable one if we add more devices to spread all around the house. Finally, the subscriber will receive the notification from both devices and will inform the user through an ad-hoc user interface. The overall structure is summarized in Figure 3.

In order to avoid multiple alerts in a shorter interval of time, the notification system defines a window of five minutes. Therefore, the subscriber will inform the user about the intrusion only if the current alert is received after five minutes from the last notification sent. All the other alerts in the meantime will be discarded.

Both publishers will send a body with three parameters: timestamp, label and img_path. The timestamp will be used to keep track of the intrusion whereas the label is used also for detecting potential non-intrusions, with sounds like "Bark" and "Doorbell". Finally, in both cases we send to the user an image of the current situation at home captured through the PiCamera.

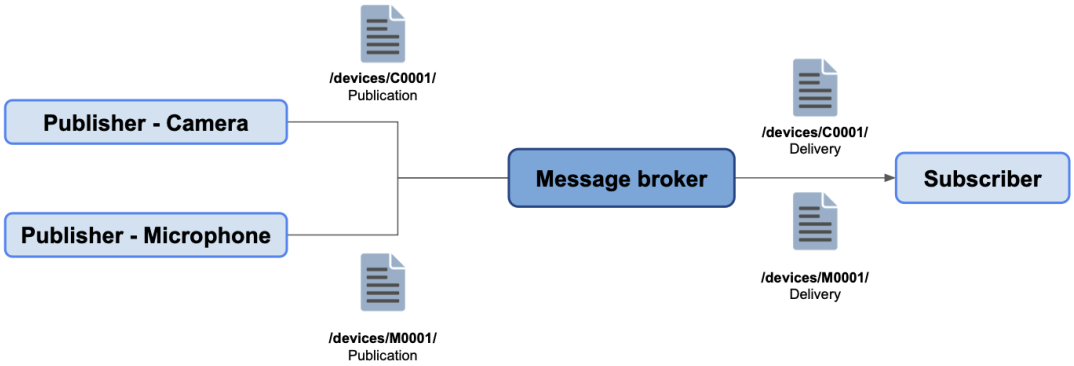

Figure 3: MQTT communication paradigm.

2.1 Storage

The camera publisher will have a trigger anytime it detects a human and all these suspicious images will be stored at 'assets/storage/images/' with the timestamp as filename in order to avoid duplicate names. However, the pipeline is a bit different for the audio classification task. Ideally, anytime a suspicious sound is detected, the system should capture an image that should be sent to the user. Since it is not possible to have two camera instances at the same time, we propose a workaround for showing anyway the current status. To be more precise, the instance of the PiCamera used for the streaming analyzes frame after frame in order to detect a potential intrusion. Therefore, we store on disk every frame under the same filename (last_image.jpg) in order to overwrite it and to not save redundant information. Therefore, every time the microphone is triggered for one of the supported classes, it will publish this given filepath under the 'img_path' attribute, that represents the actual current status at home.

3 Audio Classification

Audio classification is the task of identifying what an audio represents. This task is required in the context of a surveillance system to understand if the perceived sound is due to an intrusion. Sound events that can make you think of a danger are the glass of a window breaking, the hitting of a hammer, or a drill in action. However, very loud sounds can occur without posing a threat, think of the doorbell for example. In other cases it is important to distinguish the source: hearing the dog barking is not strange, while two people talking represents an intrusion.

In our effort to build an audio classification model, we have been inspired by the KeyWord Spotting (KWS) systems. In recent years, neural network-based systems have dominated the area and improved the accuracies of these systems. Popular architectures include standard feed-forward deep neural networks (DNNs) and recurrent neural networks (RNNs). Strongly inspired by advancements in techniques used in computer vision (e.g., image classification and facial recognition), the convolutional neural network (CNN) has recently gained popularity for KWS in small memory footprint applications. A common approach is to calculate the Mel-frequency cepstral coefficients (MFCCs) of the audio signal and to use these extracted features to train an image classification model.

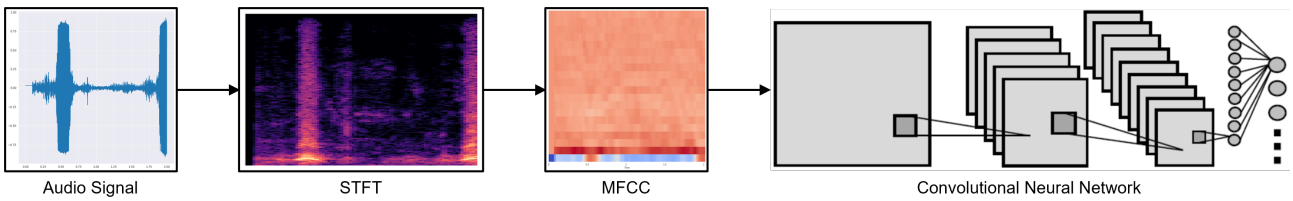

Figure 4: Feature Extraction from Audio with MFCCs.

3.1 Dataset and Pre-Processing

Our smart surveillance system is trained to recognize the sounds shown in Table 1. The cleanest and most representative samples of each class have been manually selected from FSD50K, which is a vast dataset that collects over 50,000 audio clips, the labelling of which is still in progress [6]. Since the audios from this Dataset belonging to the "Speech" class are mostly noisy, we have added three 8-minute IELTS practice listenings, which consist of fairly realistic, good-quality, and noise-free dialogues.

Table 1: Number of samples for each class.

Class	**Samples from the Dataset**	**Samples after Pre-Processing**
Bark	200	686
Doorbell	180	478
Drill	200	1442 > 400 (down-sampled)
Glass breaking	190	461
Hammer	178	1219 > 400 (down-sampled)
Speech	187 + 3 IELTS Practice Listening	1088

Having assembled this corpus of sounds, we have standardized all the files in a single format (wav), characterized by the same bit depth (32 bit), sample rate (44.1 kHz), and number of channels (mono).

To maintain a good trade-off between latency and accuracy, the classification model is trained on two-second audio. Therefore, starting from the audio present on the dataset, we derived mini-samples of two seconds sliding towards the audio with a one-second shift. The slide has been performed in order to not loose any information whereas the one-second shift has been done mainly for two purposes: having the double number of samples and for introducing an "hand-crafted" data augmentation technique. The final number of samples can be seen in Figure 5.

As a consequence, the model becomes more robust, because it is also trained to recognize the time-shift of the event. Before being fed to the model, silent samples are discarded by fine-tuning a volume threshold for each class. The number of samples obtained after this pre-processing phase is shown in Table 1.

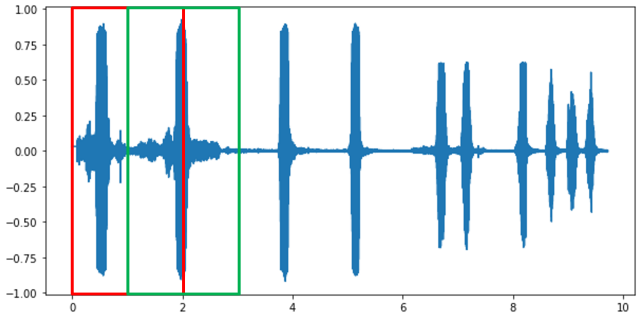

Figure 5: two-second audios are cut from the original file by progressing in one-second steps.

Several experiments were carried out to arrive at this solution. Initially, we trained the model to classify one-second sounds. However, such short audios are indistinguishable in real-time inference, especially for the "Bark" and "Speech" classes.

The dataset obtained is unbalanced. First, we trained the model on all the data and then after having performed downsampling to balance the classes, but neither of them gave satisfactory results. In this way, however, we found out that the "Bark" class tended to prevail over "Speech", while "Glass breaking" was mistaken for "Hammer" and "Drill". For this reason, IELTS practice listenings have been added to the "Speech" class, while the "Drill" and "Hammer" classes have been downsampled. Stratification has been applied during downsampling to take samples from different files in a proportional manner.

Recognizing the importance of providing the model with a large amount of data, we conducted experiments by carrying out data augmentation, using the audiomentations library for this purpose [10]. A more in-depth study should be done in the future, because our attempts have never led to real-time improvements.

3.2 Feature Extraction

Feeding raw audio file to a Convolutional Neural Network has been proved to not obtain satisfactory performances in many applications. Therefore, it is required to perform a set of pre-processing steps. First, we

moved from the time domain to the frequency domain using the using Short Time Fourier Transform (STFT). It divides a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. Therefore we end up with a two-dimensional image, for which it is possible to perform an image classification problem.

In particular with human speech recognition tasks the conversion from the Mel Spectrogram into the Mel Frequency Cepstral Coefficients (MFCCs) has been proved to be beneficial in many applications. The MFCCs produce a compressed representation of the Mel Spectrogram, focusing only the most essential coefficients, that represents the frequency ranges at which humans speak.

The hyperparameters for the feature extraction are extremely important because they have a huge impact on the model accuracy and on the inference latency. Therefore, after several tests we found out that a satisfactory trade-off among performances and model size are provided with the following ones: frame length and step of 80ms and 40ms respectively, 20 MFCCs coefficients and 32 mel bins. The audio files were kept at the original frequency of 44.1kHz. Therefore resampling at a lower frequency has not been performed in order to preserve as much information as possible.

3.3 Architecture

Recently, depthwise separable convolution has been proposed as an efficient alternative to the standard 3-D convolution operation and has been used to achieve compact network architectures in the area of computer vision [9]. DS-CNN first convolves each channel in the input feature map with a separate 2-D filter and then uses pointwise convolutions (i.e. 1x1) to combine the outputs in the depth dimension. By decomposing the standard 3-D convolutions into 2-D convolutions followed by 1-D convolutions, depthwise separable convolutions are more efficient both in number of parameters and operations, which makes deeper and wider architecture possible [17]. In this work, we have adopted a depthwise separable CNN based on the implementation of MobileNet [9], as shown in Figure 6. An average pooling followed by a fully-connected layer is used at the end to provide global interaction and reduce the total number of parameters in the final layer. While MobileNet has 13 depthwise separable convolutional layers, our model requires only two to achieve satisfactory performance on the task, both in terms of accuracy and size.

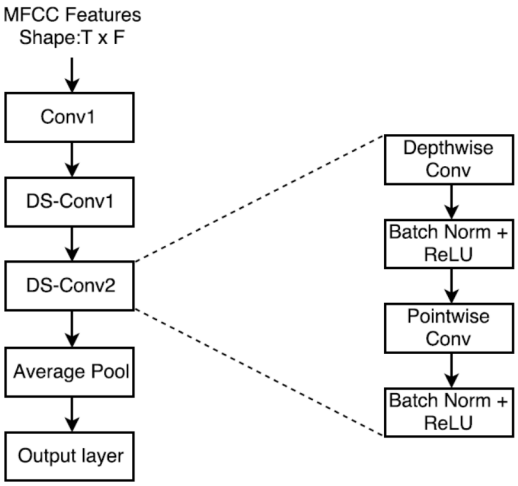

Figure 6: Depthwise separable CNN architecture of MobileNet.

3.3.1 Benchmarks

We conducted several experiments before arriving at the aforementioned architecture. Besides testing several famous architectures, we also tried to fine-tune pre-trained audio classification models, as well as widen and deepen ours. All our benchmarks have been conducted with the previously described training set and with an external independent validation set, manually generated with audio recordings from YouTube videos. We balanced the number of samples for each class, in order to compute the accuracy score of each architecture that have been reported in Table 2. We did not considered a proper "test set" for a final benchmark because we directly tested in real time with the audios recorded from different sources through the microphone.

YAMNet is a pre-trained deep neural network that can predict audio events from 521 classes [16]. The model uses the original MobileNet architecture (13 hidden layers) and was trained using the AudioSet-YouTube corpus, which gathers 2.1 millions audio files. We used this model as an high-level feature extractor and then fed the obtained audio files embeddings to a shallow neural network to classify our classes. We decided to consider

Table 2: Performance of different architectures.

Architecture	Accuracy	Model Size
DS-CNN (from Lab3)	77.78%	566 kB
VGGish [8]	77.50%	18.0 MB
YAMNet (Fine-Tuning) [16]	80.62%	15.4 MB
Music Tagging CNN [3]	73.89%	1.8 MB
MobileNet (13 layers) [9]	70.83%	13.9 MB
MobileNet (3 layers)	77.64%	126 kB
MobileNet (2 layers)	75.56%	54 kB
MobileNet (2 layers, $\alpha = 2$)	80.00%	186 kB

it as our upper-bound performance benchmark and the final selected architecture for the audio classification task was the MobileNet with only two hidden layers and the double number of neurons on each layer ($\alpha = 2$).

3.4 Model training and Hyperparameters

Each model has been trained for 20 epochs using the ADAM optimizer. The default learning rate (0.001) is maintained for the first ten epochs, and decreases exponentially thereafter. Magnitude based structured pruning has been applied to optimize the models, setting the initial sparsity to 0.3 and applying PolynomialDecay from the 5th to the 15th epoch, with a final sparsity of 0.4. Moreover, the final size of the models is further reduced by applying weights-only post-training quantization. Finally, the models have been compressed to actually see the benefits of pruning.

After having acknowledged the potential of shallower MobileNets, we have made experiments trying to widen them. By doubling the number of nodes in each layer, we got better results in terms of accuracy, but the size of the network has increased dramatically. As already mentioned, the best compromise was reached with two depthwise separable layers and with twice the number of nodes in the layers.

3.5 Inference

The microphone connected to the raspberry will be always awake and its first goal is to detect any noise. Therefore, it will record in loop just tiny chunks of audio and only whenever a given volume threshold has been crossed, it will record a $2s$ audio. This threshold has been properly tuned and we found out that based on the background noise on our houses, a good trade-off was around 1000.

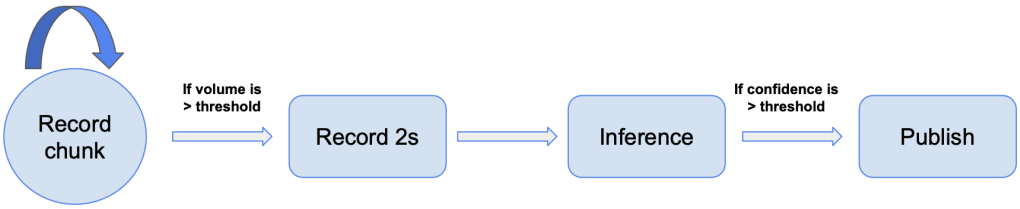

Figure 7: Real time audio inference pipeline

Then, once a sound has been detected, the microphone records the next two seconds. This audio will be preprocessed in order to extract its spectrogram and finally it will be given as input of our trained Deep Learning algorithm. Finally, the algorithm will predict its class and if the confidence for the current prediction will be above a threshold of 0.5, a notification will be published on '/devices/M0001'. The complete pipeline is graphically described in Figure 8.

4 Human Detection

4.1 Feature Extraction

The PiCamera is constantly available and it checks frame by frame the presence of any human. Anytime the algorithm detect a human, it will write on disk the picture and it will finally publish the message under the MQTT framework. For this task we did not trained any deep model, but we used OpenCV's HOGDescriptor

(Histogram of Gradients) in order to extract the features from the image. It has been introduced by Navneet Dalal and Bill Triggs [4] and it focuses on the structure or the shape of an object, breaking down the image into smaller regions and for each region, it calculates the gradients and orientation.

4.2 Algorithm

The algorithm used by OpenCV is the Linear SVM classifier, to which we set the coefficients of the classifier trained for people detection [5]. The inference is performed through OpenCV's 'detectMultiScale' method, that detects objects of different sizes in the input image and therefore it will returns a list of rectangles. The algorithm was fed with the grey image, in order to highlight the contrast of the picture and to improve the detection.

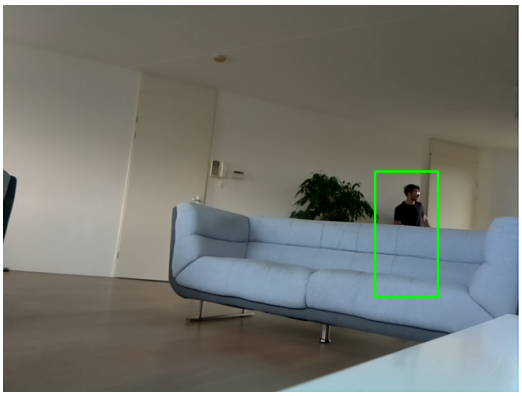

Figure 8: Human detection

Two hyperparameters have been tuned: winStride and scale. The former dictates the "step size" in both the x and y location of the sliding window. This sliding window will be used to iteratively extract the HOG features from the image and to give them as input of the SVM classifier. Obviously, it is important to find a good trade off among performances and sizes, because the smaller the window will be, the slower the model will be as a consequence. Therefore, after an in depth analysis, the final value was '(10,10)'. Finally, the second hyperparameter was the 'scale' that controls the factor in which our image is resized at each layer of the image pyramid (due to the multi-scale inference), ultimately influencing the number of levels in the image pyramid. Here, the chosen value was 1.01 (default one) because it allowed to have much less overlapping bounding boxes. An example of human detection can be observed in Figure 7.

4.3 Post processing

The scale factor of the inference pipeline allowed to reduce the overlapping boxes, but they were not yet completely removed. Therefore, an ad-hoc post processing of the resulting bounding boxes appeared to be necessary. Hence, in order to remove the remaining overlapping bounding boxes the Non-Maximum Suppression algorithm [13] has been performed, a fairly well known technique for filtering the predictions of object detectors. It requires all the predicted bounding boxes and an overlapping threshold. Very briefly, the algorithm will select the predictions with the maximum confidence and suppress all the other predictions having overlap with the selected predictions greater than the given threshold, that is 1.00 on our case.

5 User Interface

For the purpose of making the interaction between our application and the user as easy as possible we decided to implement a Telegram bot, in which by using simple commands, it is possible to interact with all the functionalities that our system offers. Once the subscription to the application has been finalized, the chat_ids of the people in the house can be inserted into the system, in order to limitate the service only to the owners. After that, the user has to message @DomesticSounds_bot in order to setup the account. First, he has to activate it through the command *"\start"* and then he can choose among the different functionalities:

1. *"\enable"*: activate intrusion notifications. If the notification are already enabled, it displays an error message to remind it to the user.
2. *"\disable"*: stop intrusion notifications. If the notification are already disabled, it displays an error message to remind it to the user.

3. *"\report"*: returns the list of past intrusions.
4. *"\help"*: return the list of the contact of the creators, in order to get help or give some feedback.

Figure 9: Brief overview of bot's commands

Once the notifications have been enabled, every time the system finds a potential danger through audio or video, a message containing the image and the class of the danger is sent to warn the user that something is happening in the house.

5.1 Implementation

We used the telegram.ext package [15], and in particular *telegram.ext.handler* to manage commands and make them responsive directly in the chat.

As concerns the intrusion message, we used The Bot API [14], which is an HTTP-based interface that through a synchronous function can send messages to the user. In particular, the main method we used is:

- **sendPhoto**(chat_id, photo, caption = NULL, disable_notification = FALSE,reply_to_message_id = NULL, reply_markup = NULL, parse_mode = NULL): chat_id is the identifier of the user and it is inserted during the initial registration phase, photo contains the file showing the possible danger, and then we wrote in the caption the class associated with the potential risk detected.

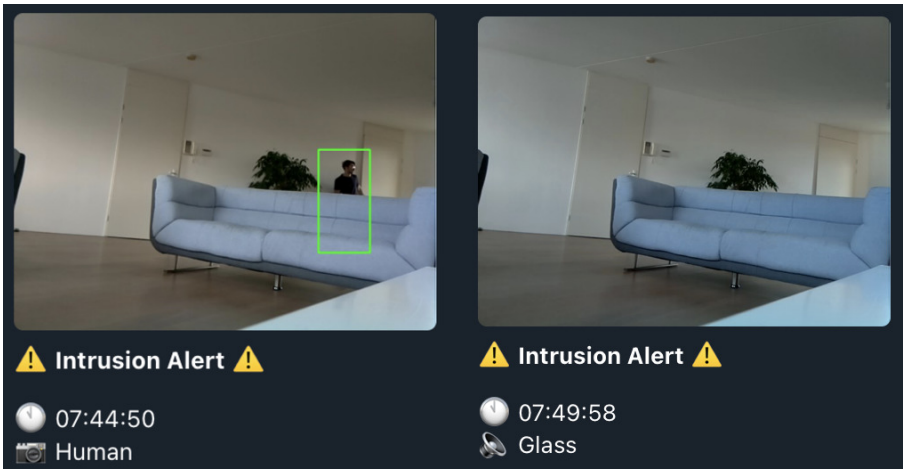

Figure 10: Example of notifications from camera and microphone

6 Limitations and further improvements

This project lays the foundations for a very broad and interesting domain and applications. Even though the obtained results are considerable, many limitations reduced the possibility of achieving an even better accuracy. Therefore there is still room for improving the current performances, making this service definitely more robust. In particular the criticism rely on three main areas: classification task (dataset and deep learning algorithms), quality of the hardware and storage.

6.1 Classification task

Following the limitations described above, a very first big improvement may rely on the usage of more classes for both microphones and camera. Thinking about the microphone, other classes like car, thunder, footsteps and similar ones may make the audio classification task way more robust. On the other hand, thinking about the camera, in this project we did not train an ad-hoc neural network but we only relied on an OpenCV pre trained model. Therefore, a first improvement may require the training of an ad-hoc neural network, taking into account all the edge cases of an intrusion like low light conditions, dark clothes and so on. Moreover, having the possibility to detect also "neutral" people, like the house owners and domestic animals would allow to keep the camera active all day long.

6.2 Hardware

Moving the discussion to the hardware, obviously using better cameras and microphones could be extremely beneficial for making accurate predictions. In fact, the sounds recorded from the microphone are well recognized only if the microphone is close to the source. In addition, cameras with night vision and a significant better quality could improve the computer vision task.

Moreover, one may also think to create an ecosystem of devices like speakers and lights that can be turned on in case of an intrusion in order to scary in in a rudimentary way the intruders.

6.3 Storage

Finally, the last aspect of the project that could be improved regards the management of the saved images of the potential robbers. At the moment all the images captured with the PiCamera are stored on the Raspberry, but due to its limited memory, it will be saturated in a while, especially with many cameras installed. Therefore, having access to a cloud to which can periodically transfer these images can be a strong improvement and it would definitely make the overall system more robust and resilient.

References

[1] ISO/IEC PRF 20922. Mqtt. https://mqtt.org.

[2] Amazon. Amazon's Alexa Guard Technical Specifications. https://www.amazon.com/gp/help/customer/display.html?nodeId=G8W367YLULTRSB2S.

[3] Keunwoo Choi and Johan Pauwels. Music Auto-Tagger. https://github.com/keunwoochoi/music-auto_tagging-keras.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[5] OpenCV Docs. Cv::hogdescriptor struct reference.

[6] Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra. Fsd50k: an open dataset of human-labeled sound events, 2020.

[7] Google. Google Nest Cam 4b Technical Specifications. https://store.google.com/it/product/nest_cam_indoor_specs?hl=it.

[8] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron Weiss, and Kevin Wilson. Cnn architectures for large-scale audio classification. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. arXiv, 2017.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[10] Iver Jordal, Araik Tamazian, Emmanouil Theofanis Chourdakis, Céline Angonin, askskro, Nikolay Karpov, Omer Sarioglu, kvilouras, Enis Berk Çoban, Florian Mirus, Jeong-Yoon Lee, Kwanghee Choi, MarvinLvn, SolomidHero, and Tanel Alumäe. iver56/audiomentations: v0.24.0. https://doi.org/10.5281/zenodo.6367011, March 2022.

[11] The motionEye Project. Motioneyeos. https://github.com/motioneye-project/motioneyeos/wiki.

[12] Raspberry Pi. Raspberry Pi 4b Technical Specifications. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[13] Jatin Prakash. Non maximum suppression. https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/.

[14] Telegram. Telegram Bot API. https://core.telegram.org/bots/api.

[15] Telegram. Telegram.ext package. https://python-telegram-bot.readthedocs.io/en/stable/index.html.

[16] TensorFlow. Sound classification with YAMNet. https://www.tensorflow.org/hub/tutorials/yamnet.

[17] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers, 2017.