Internet of Things: Spy Your Mate

Giovanni Baccichet
10851745

Politecnico di Milano — June 29, 2022

1 Introduction

In this project, you are requested to analyze the traffic generated by a webcam call and, from such traffic, identify whether a person or only the background is recorded in the video. You have to approach this problem from a *Machine Learning* (ML) perspective.
It is important to point out the hardware and software (specific versions) used to fullfil the requirements, in order to replicate the achieved results. For this project I used:

- Apple `MacMini8,1` w/ Intel Core i5 and 32 GB RAM, running `macOS 12.4`;
- Wireshark `3.6.6`;
- Jupyter Notebook w/ Python `3.9.12`, using `Anaconda-Navigator`;
- NodeRED `2.2.2` running on Docker `4.9.1`;
- Zoom.us `5.10.7`.

Everything disclosed in this report (except for the network dumps, due to privacy reasons) can be found in this repository: 🔗 github.com/GiovanniBaccichet/IoT-human-detection.

2 Network Setup

Figure 1 represents a simplified version of the network topology used. Briefly: two computers connected to the same network, the red one using a 5 GHz WiFi connection, the other one using a Gigabit Ethernet connection. In the following section the *Internet* portion of the diagram will be addressed, along with the network traffic generation process followed during the experiment.

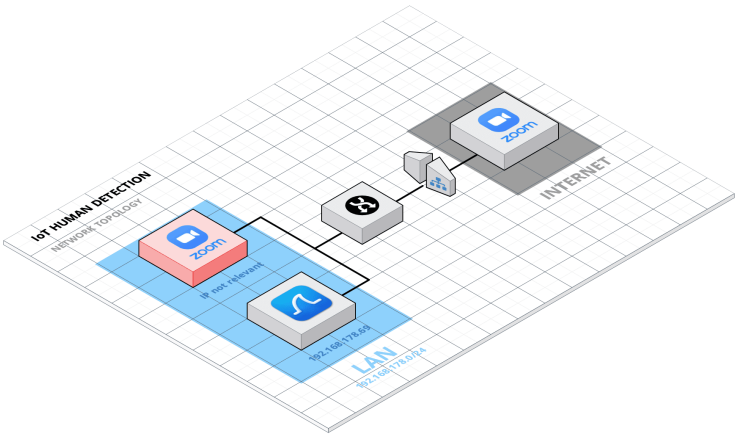

Figure 1: Network Topology: a simplified version of the LAN, target in red.

2.1 Traffic Generation

Traffic generation was performed using the PC dumping the network traffic as a participant of a Zoom call, in which the target PC was the administrator and had the webcam turned on. The IP address of the eavesdropper was `192.168.178.69`. Since Zoom is not a *Peer-To-Peer* application, an IP was required to clean out the traffic from other unwanted destinations (sources). For this purpose an application firewall (`Little Snitch 5.4.1`) was used: it allowed to visualize a list of addresses used by Zoom to relay the traffic, see figure 2. This allowed to get an idea of the range of IP addresses that the service provider used for allowing calls between users.
The time period in which someone was framed by the webcam was about 20 minutes, the same for the background only, for a total of 40 minutes.

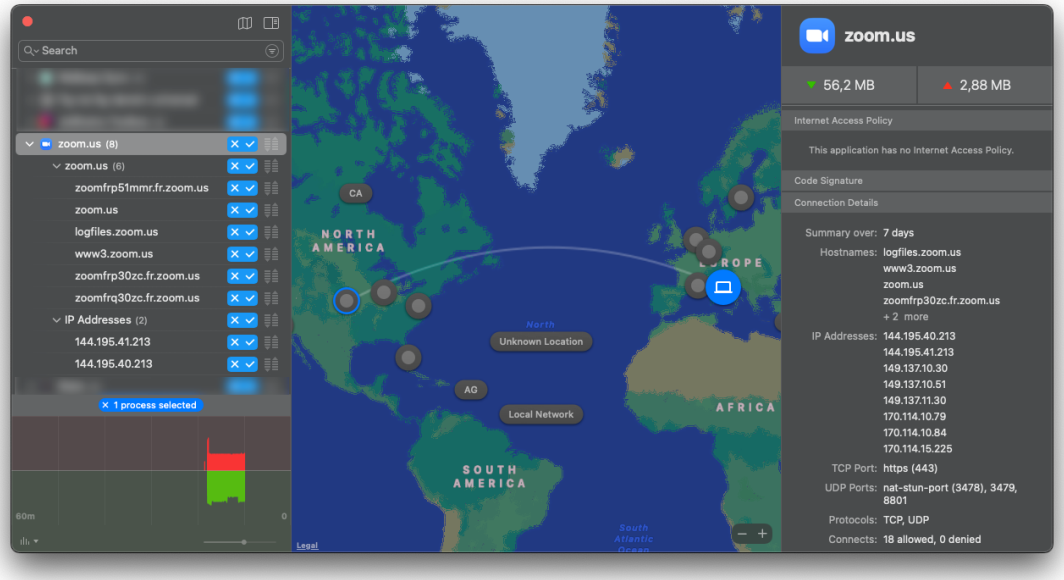

Figure 2: Little Snitch instance, visualizing Zoom.US traffic (*IP addresses don't match to the Juputer Notebook's ones because this screenshot was taken in a second moment*)

2.2 Traffic Acquisition

Traffic acquisition was performed using `Wireshark` to generate a *Packet CAPture* (PCAP) file, from which some interesting information was extracted and exported in *Comma Separated Value* (CSV) format, using `tshark`. The latter is presented in a terminal fashion, and the command used was:

Command Line

```
$ tshark -r "dataset-zoom-us-20+20_min.pcapng" -T fields -e
frame.time -e ip.src -e ip.dst -e ip.proto -e frame.len -E header=y
-E separator=, > "zoom_dataset.csv" -Y 'ip.addr == 149.137.12.48'
```

Said command allowed to export a CSV with the following characteristics (columns):

- Timestamp;
- Source IP address;
- Destination IP address;
- Protocol;

- Frame Length.

From an initial network dump of `8.43 GB`, this additional step allowed to narrow down to only `10.8 MB`.

3 Node-RED Setup

In order to acquire a *Ground Truth*, and represent it, a *Node-RED* dashboard was used. In the figures below it is possible to see the *flow* used (Figure 3), and the *Ground Truth Builder* widget (Figure 4).
Node-RED was running in a Docker container: so the CSV generated and (at the end) parsed were inputted using a remote shell connection.

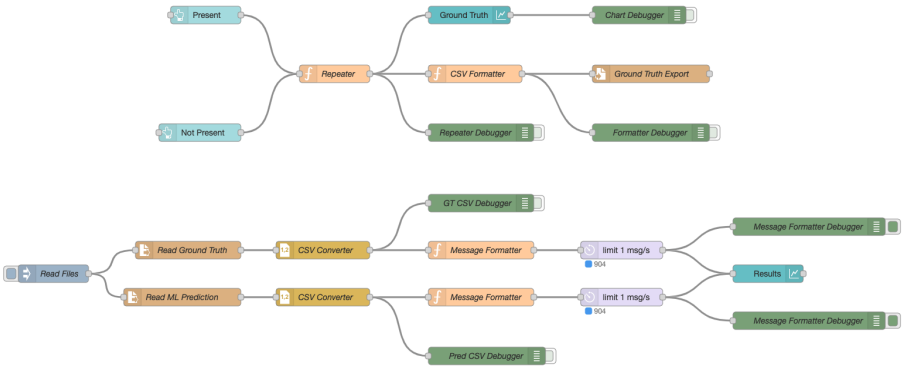

Figure 3: Node-RED Flow

The ground truth consisted, as mentioned above, in a CSV reporting the `timestamp` and the value `1` if someone was framed, otherwise `0`.

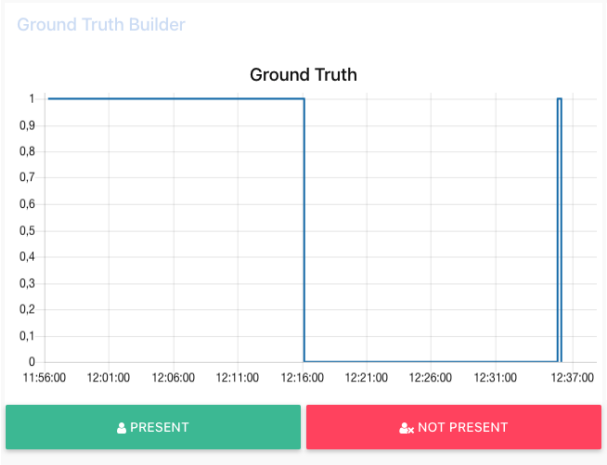

Figure 4: Node-RED Ground Truth Builder

4 Features and Data Visualization

Everything disclosed in the previous section was used as an input for the Jupyter Notebook (which can be found in the repository mentioned in the Introduction). The most technical details won't be addressed in this report, since every step is commented in the code itself.
On the other hand, in this section some remarks about the data acquired (before the model training) will be given.
The approach used to select the features was incremental: starting with few features, visualizing them graphically and trying to train the model. The features used were: `PacketNumber` (*number of packets within*

a time interval of 450 ms), `PacketIntervalAvg` (*average amount of time between packets within 450 ms* - figure 6), `PacketSizeAvg` (*average packet size* - figure 8), `PacketSizeStd`, `PacketSizeMed`, `PacketSizeMin`, `PacketSizeMax`, `InboundCount` (*number of packets received by the eavesdropper* - figure 9), `OutboundCount` (*number of packets sent by the eavesdropper*).
The correlation matrix between said features can be seen in figure 5.

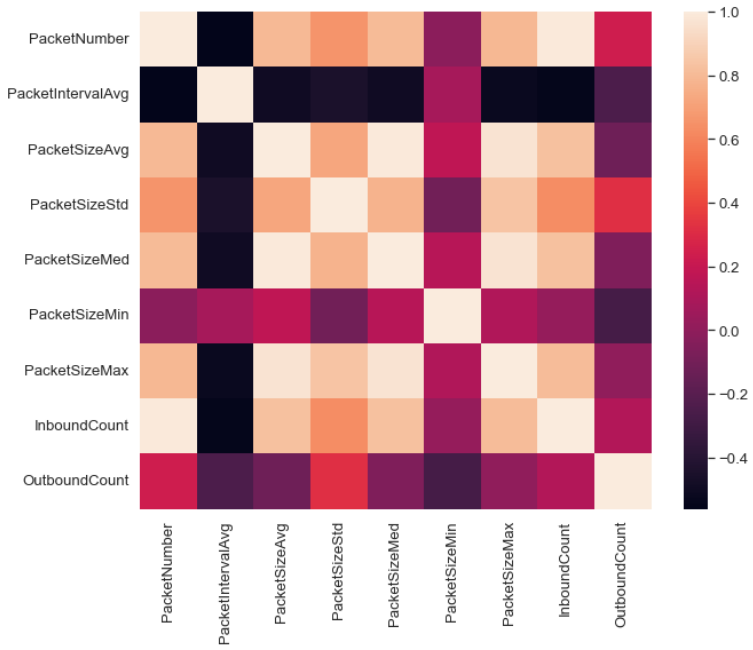

Figure 5: Selected features correlation matrix

Plotting some of those features, with respect to the ground truth (pink line), it is evident when someone if framed and when not, despite the fact that some features are correlated (not completely) with some others.

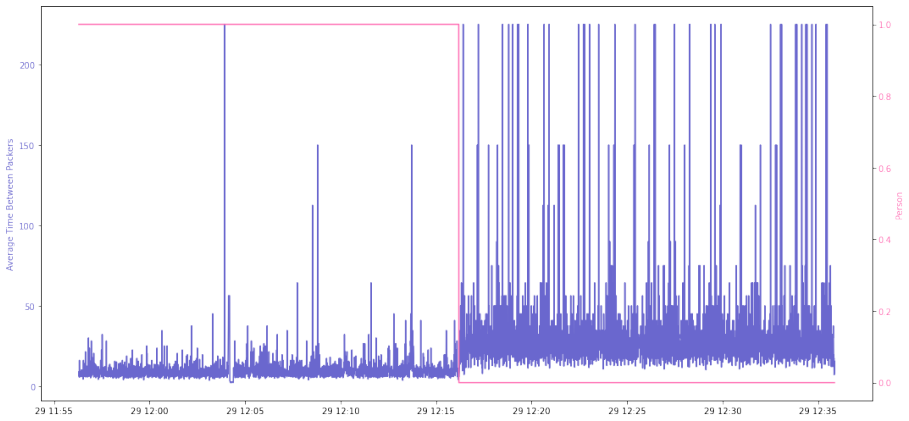

Figure 6: Average Time Between Packets

In the Jupyter Notebook are also attached some other diagrams aimed at visualizing the data frame we are dealing with and also the selected features and better understanding their relevance, from a macro standpoint.

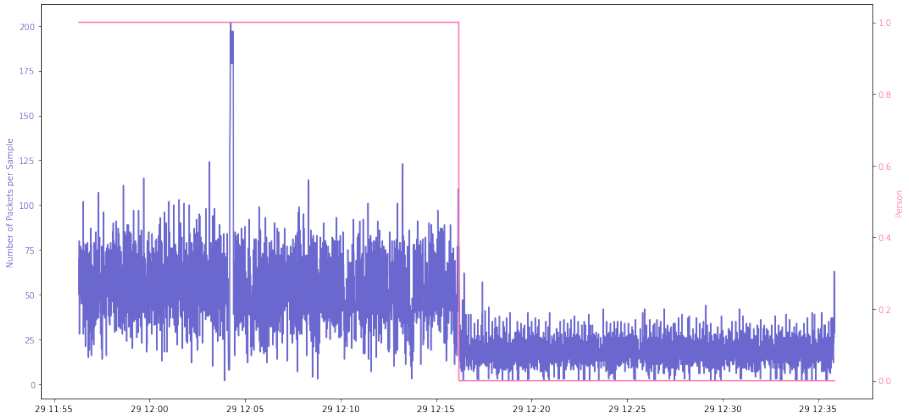

Figure 7: Packets per Sample of 450 ms

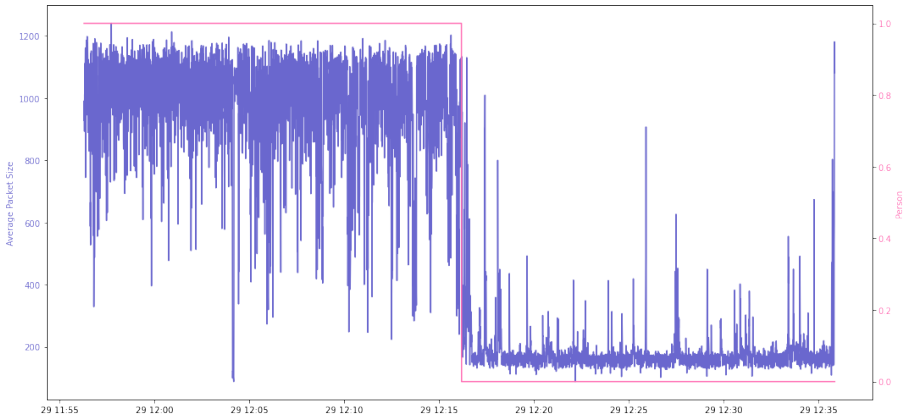

Figure 8: Average Packet Size within a Sample of 450 ms

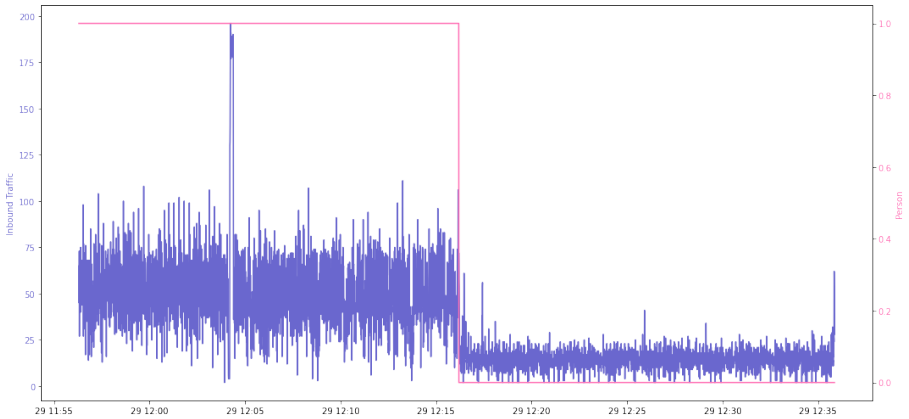

Figure 9: Inbound Traffic

5 Results

As anticipated by the data visualization, the results were very accurate. The ML algorithms tested were: *Random Forest*, *K-Nearest Neighbor*, *Support Vector Machine*, *Logistic Regression*.
Every algorithm tested presented very similar results, with the lowest accuracy reported of around 97%, and the highest one of 99%. Everything is reported in the Jupyter Notebook, however in this section will be included some more information about only 2 of the trained models since they all behaved the same.

5.1 Random Forest

Label	Precision (%)	Recall (%)	F1 Score (%)
`Background (0)`	99.6047	97.2973	98.4375
`Person (1)`	97.4122	99.6219	98.5047

Table 1: *Random Forest* trained model metrics

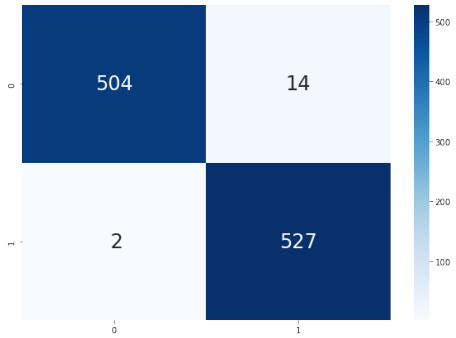

Figure 10: Random Forest Model Confusion Matrix

5.2 Support Vector Machine

Label	Precision (%)	Recall (%)	F1 Score (%)
`Background (0)`	97.5285	99.0347	98.2759
`Person (1)`	99.0403	97.5425	98.2857

Table 2: *Support Vector Machine* trained model metrics

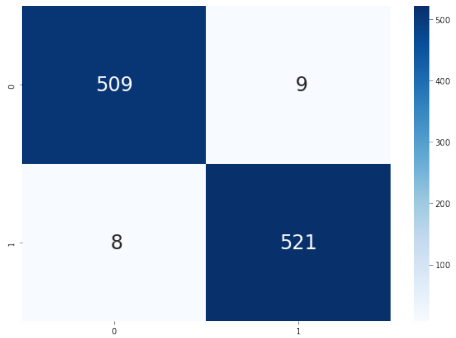

Figure 11: Support Vector Machine Model Confusion Matrix

5.3 Predicted vs Actual Data Visualization

Taking as an example the RF model, the actual predicted vs actual values chart looks something like figure 12, and the SVM one, like figure 13: this is completely unreadable from this report. In the Jupyter Notebook the plot can be enlarged and inspected, the values match almost completely; this can be noticed in figure 14, an enlargement of figure 12, with some line tweaking to make the visualization easier.

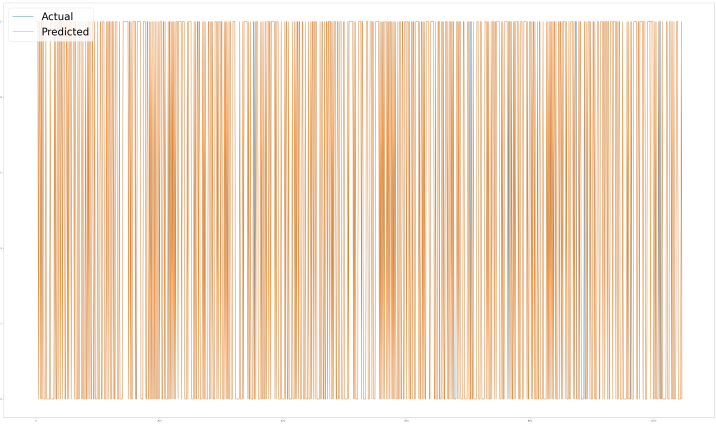

Figure 12: Random Forest Model Predicted (orange) vs Actual (blue) Values

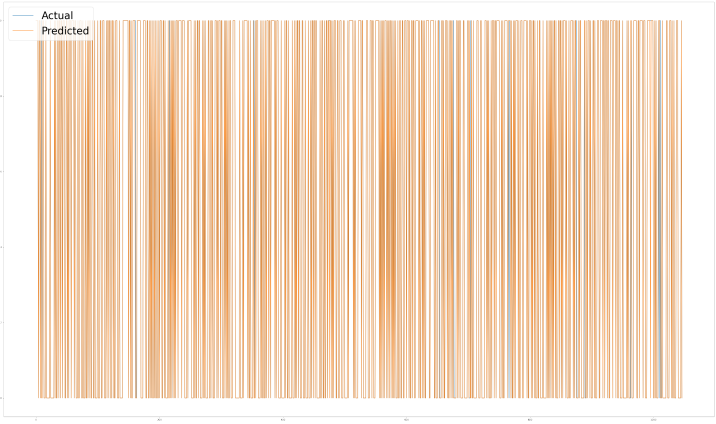

Figure 13: Support Vector Machine Model Predicted (orange) vs Actual (blue) Values

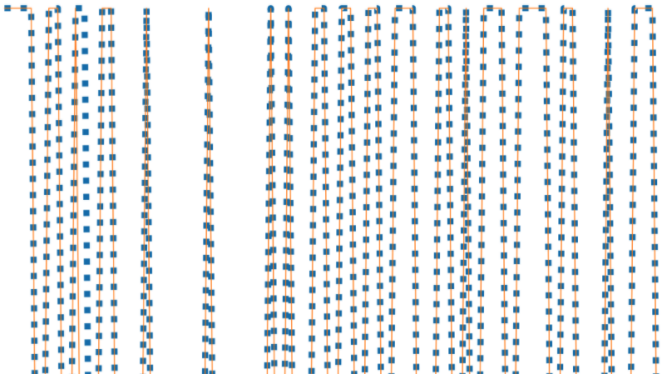

Figure 14: Random Forest Model Predicted (orange) vs Actual (blue) Values - Enlarged (*as expected from a 99% accuracy, the lines overlap almost every time*).

5.4 Node-RED Results Chart

Results were exported from the Jupyter Notebook in CSV format, then copy-pasted in the Node-RED's docker container, in order to be visualized using the dedicated dashboard. Since the procedure would have been the same for every algorithm, and so the results, only the RF model ones were plotted using this application.

Figure 15: Node-RED ML Results diagram.

In order to better visualize the data provided by the Jupyter Notebook, the Node-RED chart was realized using 2 `delay` nodes (one for the ground truth, the other one for the predicted values): this allowed to spread data (also having a dynamic view of it) in time and not having a cluttered diagram with all the CSV data all in one place at the same time, resulting in being unreadable.
As disclosed in the previous section, due to the accuracy of the model, the lines of actual and predicted values overlap almost every time; in the screenshot displayed in figure 15 it is possible to see one time that the model got the wrong prediction.