Object detection in the context of intrusion detection

Timon Zimmermann
Swiss Federal Institute of Technology
timon.zimmermann@epfl.ch

Akshat Dewan*
Swiss Federal Institute of Technology
akshat.dewan@epfl.ch

Bixio Rimoldi†
Swiss Federal Institute of Technology
bixio.rimoldi@epfl.ch

June 13, 2017

Abstract

This paper presents the implementation of a "dangerous object" detector with its entire pipeline from the capture of the images to the triggering of a light or alarm in the case of a detection. With the help of current advances in machine learning, we use an efficient neural network architecture specifically made to keep the processing fast while maintaining an accuracy competing with state-of-the-art results. At the end, we propose a solution which processes images fast-enough, while maintaining a high recall and precision for competent authorities to be contacted in the case of an alarming detection.

I. Introduction

Object detection algorithms have been around for a long time and are a huge challenge in *computer vision*. Indeed, the ability to accurately and efficiently detect objects within a scene have many applications. During this project we investigated the problem in the context of intrusion detection, particularly the localization and recognition of dangerous objects on image coming from security cameras.

Furthermore, one way to reducing this kind of infringements is the prevention via a prompt and automated detection of such intrusions, so that competent authorities are able to react and take care of the matter immediately. Although solutions to this problem have been studied for a long time, using traditional approaches from computer vision coupled with basic machine learning techniques [1, 2], in the last couple of years with the ever growing enthusiasm for machine learning applied to images, particularly Convolutional Neural Networks, we have seen exceptional improvements in term of efficiency and accuracy.

Keeping in mind the final implementation, a real-time prototype fully working with our in-lab camera, we had to keep the inference time for each frame as low as possible. To do so, we adopted a special type of Convolutional Neural Network specifically made to keep the processing fast while maintaining an accuracy competing with slower state-of-the-art object detector. This kind of neural network, called R-FCN [3], has been introduced in a paper to demonstrate how any convolutional object classifier can be *effectively* converted to an object detector. Drawing its inspiration from previous advances [4, 5] to make neural networks more efficient, with regard to the inference time and precision.

Detecting particular objects with CNNs from security camera faces several additional challenges:

*Supervisor
†Professor

- Acquiring and labelling our own dataset for a specific type of objects is long and fastidious, it takes a lot of man-hours.
- To properly train such deep networks, which contain millions of parameters, you need a very large amount of data. To make this problem solvable, the usual solution is as follow:
  - Load pretrained weights (parameters) for the backbone CNN from an open-sourced database, ImageNet in our case (1.28 million of images for over a thousand classes).
  - Fine-tune the weights of the architecture with our own "smaller" dataset of special classes.
- As previously mentioned, requiring the camera stream to be processed in real-time, or nearly, adds a layer of complexity in that it is hard to find a good balance between accuracy and speed.
- Security cameras do not offer good quality images, thus making it harder for the backbone CNN to extract clear and meaningful features which will later be used by the layers in charge of the object detection.

Lastly, as it will be discussed in section ii, we opted for 3 types of object to detect: guns, knives and masked persons. Given these points, we present in this paper the implementation of a "dangerous object" detector with its entire pipeline from the capture of the images to the triggering of a light or alarm in the case of a detection.

II. Related Work

i. Intrusion detection

Totally preventing security breaches with the existing techniques and security technologies is unrealistic. Indeed, many intrusion detection systems (IDSs) are rule-based, which makes them obsolete in face of new type of intrusions. As a result, recent advances in machine learning and algorithms provide new frameworks to tackle the problem by generalization. In other terms, such algorithms try to answer the question "What is an intrusion?" rather than memorizing patterns from previous attacks by heart. This novel approach of questions allows adaptive solutions and therefore more *intelligent* and *elaborated* systems.

However, including all types of malicious activities under one generalizing concept also seems unrealistic. Hence, to make the problem more concrete, one can focus on characteristic parts of such intrusions. For this reason, many current systems using machine learning are focusing, for example, on detecting guns [6], masked persons [7].

In the context of this project, we decided to opt for a similar technique as discussed in ii. Namely, focusing on particular objects which are symptomatic of ongoing illicit activities such as guns, knives and masked persons. From there, the idea is to imply that detecting such objects is a sub-problem of scene understanding in the context of intrusions, but sufficient in most of the cases to keep the alarm miss-rate low.

ii. Object detection

The problem of object detection consists of two main parts: accurately distinguish objects' position from the background and then classify them into one of the $N+1$ classes (N relevant classes to the application and one for background, or "other") we are interested in.

To do so, such algorithm can be categorized from the technique they use to *propose* such regions, or objects' position. Indeed, some rely on ad-hoc techniques to extract regions of interest which are then passed through a simple classification algorithm or neural network. On the other hand, since this technique suffers from its inability to train the region extractor and the classifier together, others try to put everything in a single unified network.

Ad-hoc approach Such algorithms, as explained above, consists of two completely separated systems:

1. An object proposal method (called "detection proposals algorithm") [8, 9]

2. An object classifier algorithm, fed with regions extracted from the previous system

Now, even if each part individually performs well at their own task, it does not imply that they will work in harmony. Indeed, stacking systems one behind the other has the tendency to exacerbate the error of the preceding algorithm, making the error drift from end-to-end. Additionally, each system including its own overhead, they are usually much slower assembled together than solely one full-blown architecture.

Last but not least, adding to the point about their slowness, such region proposal techniques, for the most part, operate on the entire image. Hence increasing their computational complexity.

These systems, in conjunction with a neural network taking care of the object classification, have been introduced in 2015 [4] for object detection.

"Region proposal networks" approach Introduced later in 2005 [5], *Region Proposal Networks* are used to decide "where" to look for the sake of computational requirements. Instead of considering all the possible windows from the original input image as potential candidates, it quickly scans every locations from the feature map in order to output k bounding boxes with its corresponding score representing the probability to contain an object, thus assessing whether or not further processing should be carried out with this region. Namely, feeding the cropped zones into a classifier network.

Such architectures work as follows:

1. A backbone stack of convolutional layers produce a feature map from the input image.

2. Two different heads (or sub-networks) emerge from this feature map: (1) the RPN (Region Proposal Network) and (2) a straight copy of the feature map.

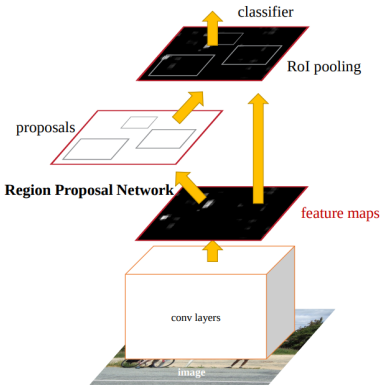

Figure 1: *Faster R-CNN architecture, featuring the Region Proposal Network (RPN)*

3. The k outputs of the RPN (1) are used to perform a *Region of interest pooling* of the feature map from (2).

4. Each output of the "RoI pooling" is passed through a classifier, which can be from a Support Vector Machine, to a softmax classifier or another kind of machine learning algorithm.

This procedure is depicted on Figure 1, where each yellow arrow corresponds to a point in the previous detailed walkthrough. One important thing to notice is the total independence of the RPN to the backbone structure of convolutional layers. Especially, it means that any fully-convolutional architecture might be used, whether it be *VGG16*, *InceptionNet* or *ResNet* iv, in our case.

Another key point is the ability train them end-to-end, since the network is sufficiently continuous to use the chain-rule and permit back-propagation through the classifier, the RPN and the backbone convolutional neural network. Making the object detector and object recognizer working together toward the same objective through a unified optimization.

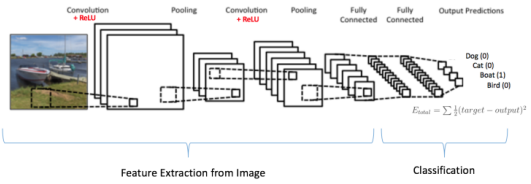

Figure 2: *Example of a CNN crafted for image classification, source*

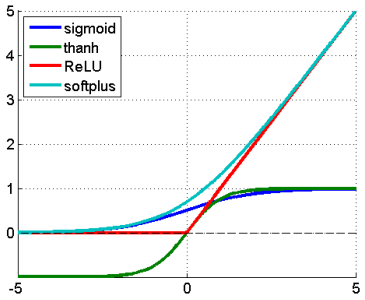

Figure 3: *Different kinds of activation functions, source*

iii. Review of Convolutional Neural Networks

Convolutional Neural Networks are a special type of *feed-forward* neural networks inspired by the visual cortex organization of animal species where the response to a visual stimuli from an individual neuron is approximated by a convolution operation. Hence, the architecture is designed to take advantage of the 2D structure and information of the input. Also important to realize, such CNNs are easier to train due to their number of parameters (or weights) which is much fewer, at equal number of hidden units, than fully connected networks. Therefore, this type of network is a very good candidate to any computer vision field, and is widely used in applications such as image recognition and classification.

Last but not least, as it can be seen in Figure 2, we add an additional operation called $ReLU(x) = max(0, x)$ (or any valid *activation* function, see Figure 3) to introduce non-linearity in the network. Indeed, convolutions are simply made of matrix multiplications and additions, so if we want to model some kind of non-linear real-world data, we need to account for it with such functions.

iv. Deep Residual Neural Networks

As networks get deeper, it is harder to train them but they offer better generalization capacities. As a result, recent progresses in machine learning and particularly network architecture face several challenges: vanishing gradients, exploding gradients, weights initialization, and so on.

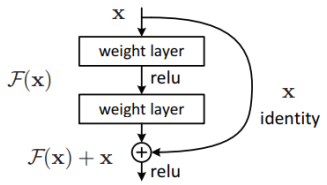

Figure 2. Residual learning: a building block.

Figure 4: *Residual learning novel building block, from [10]*

For this reason, one of the biggest breakthrough proposed lately [10] is about easing the training of substantially deeper networks. This approach is winning *1st* place in every recent image classification challenges and consist of a simple, yet very effective, solution. In detail, their discovery owes to the addition of an identity short-cut between layers, called residual learning (Figure 4) and thus leading to:

$$y = F(x, \{W_i\}) + W_s x$$

Where W_s is a projection matrix to match the dimension of x and F when the input/output number of channels are not equal.

From there the benefits are quite clear in many applications and even if the results have been empirically proven, the debate on the reasons of their efficiency is still open and there exist many possible explanations (equivalence with exponentially many sub-networks, resolving vanishing gradient...).

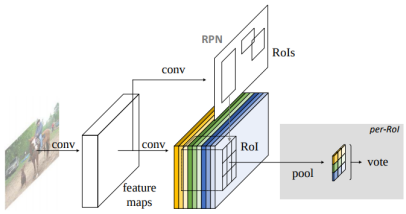

Figure 5: *R-FCN overall architecture, from [11]*

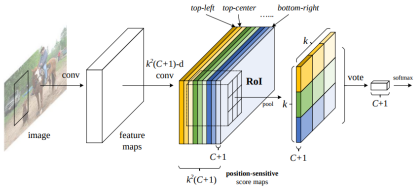

Figure 6: *Key idea of the R-FCN score map, from [11]*

III. Approach

i. R-FCN

Introduced in a paper in 2016 [11], the idea is to demonstrate how any convolutional object classifier can be effectively converted to an object detector. To achieve this goal, they replaced the "RoI" pooling layer by a bank of position-sensitive score maps, solving the dilemma between translation-invariance for object classification and translation-variance for the detection part. Leaving the backbone sack totally independent of the RPN and the score maps, we opted for the aforementioned "Deep Residual Neural Networks" with 50 layers (ResNet-50) in this project for their outstanding performances on various computer vision tasks, which we pre-trained on ImageNet to bootrap our model and then fine-tune on our own dataset, as discussed in subsection ii.

Their method, inspired from *Faster R-CNN*, naturally adopt any fully convolutional network as a backbone and obtains competitive results on PASCAL VOC while achieving inference-time 2.5-20x faster than its competitors.

The architecture is quite similar to the one of Faster R-CNN and depicted on Figure 5. Actually, the only difference is in the second sub-network, which was previously a copy of the feature map, but now is a computation of the position-sensitive score maps. Therefore, the "RoI pooling" will be computed on these score maps.

The details on how this bank of score maps work are in Figure 6, where k is the discretization of each region of interest (typically $k = 3$) and C represents the object categories we are interested in. Hence, producing an output channel of size $k^2(C+1)$, the k^2 meaning a kxk spacial grid describing relative positions (top-left, top-center, ..., bottom-right). The score for each (i, j)-th bin with $0 \leq i, j \leq k-1$ is computed with the following formula:

$$r_c(i, j|\theta) = \sum_{(x,y)\in bin(i,j)} s_{i,j,c}(x + x_0, y + y_0|\theta)/n$$

Thus $r_c(i, j)$ is the response of the bin i, j for the c-th category, since $s_{i,j,c}$ is the score map of one particular category, (x_0, y_0) the top-left coordinate of a given region of interest and θ represents all the learnable parameters of the model.

From there, one has many choices but in the original paper they adopted the technique to average the scores over the all (i, j) bin:

$$r_c(\theta) = \sum_{(i,j)} r_c(i, j|\theta)$$

Which produces a vector of dimension $C+1$, one value for each category's response. Finally, this same vector is passed through a softmax, over the categories:

$$s_c(\theta) = e^{r_c(\theta)} / \sum_{\tilde{c}=1}^{C+1} e^{r_{\tilde{c}}(\theta)}$$

This last $C+1$-dimensional vector of softmax responses is then used to compute the cross-entropy loss during training and to rank the different region proposals for the inference. This whole process is summarized on Figure 7, when trying to correctly detect the baby on the picture, and on Figure 8 is a concrete case of

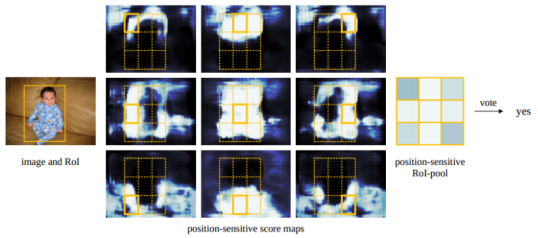

Figure 7: *Visualization of the score maps for $k^2 = 3x3$, for category* person*, from [11]*

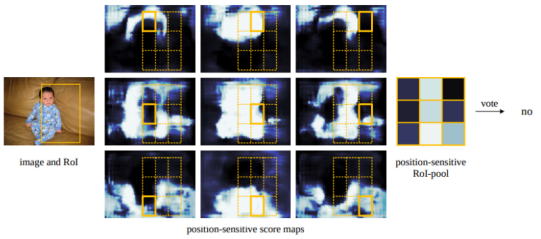

Figure 8: *Same visualization as Figure 7 but when the "Region of interest" does not correctly overlap the object, from [11]*

respecting the translation-variance of an object detector, i.e. we do not want to accept objects within a bounding box that does not properly overlap it.

Given these points, we used the neural network in the famous deep learning framework called "caffe". Although we could have implemented the whole thing by ourself, we chose not to re-invent the wheel and therefore forked the non-official implementation of the original paper [1], then modified it to our needs.

ii. Datasets

In front of the compelling evidence that resources for target classes, namely guns, knives and masked persons, are so scarce, we decided to build our own dataset from manually annotated videos from Youtube and LiveLeak. On top of that, we also added the few images disposing of a bounding box from an open-source database.

ImageNet and other[12] Originally created as an open-source database for object classification and not object detection, ImageNet does not incorporate a lot of images with their corresponding bounding box. Indeed, they just recently started to gather data about object positions for detection tasks and not simply classify objects.

For our case, with the three classes *guns*, *knives* and *masked persons*, the database included:

- Gun: 176 annotated images
- Knife: 387 annotated images
- Masked: 478 annotated images

So we had a dataset of 1041 images (16.9% of guns, 37.2% of knives and 45.9% of masked persons).

Later, from the realization of the short timeframe of the project we would not be able to manually annotate enough images to correctly train so many weights, we dropped the first two classes and restrained our detection to the class with the most images from our only open-source dataset: masked persons.

From there, after the first-stage's experiments, we noticed that the network was easily fooled by non-masked persons triggering a wrong detection, and hence lowering the precision. Therefore, to counteract this phenomena we decided to add one more dummy class that we would not use for the detector, but to reduce the number of false-positive: non-masked persons. Since such networks learn by examples, this is a convoluted way of showing the system what is "not" a masked person and consequently learning by *counter example* directly from the source of false-positives.

Indeed, the CNN previously had to classify (once the region is propsoed) between *masked* and *other* (or background), and thus the distinction opposing masked and non-masked persons was apparently not clear enough, as a result of the few examples we fed it with. It was drown into the *other* category. In addition, there exist many open-source datasets of annotated persons' head [2] to grab images from,

[1] `https://github.com/Orpine/py-R-FCN`

[2] `https://www.di.ens.fr/willow/research/`

thus it was easy to obtain a balanced dataset of masked and non-masked examples ($\approx$ 600 images each).

Own labelling Now that we looked for open-source databases, the only thing left was the long and fastidious labour to manually annotate videos. However, since it takes a lot of man-hours to do and considering the time left when we realized that we had to collect some more data on top of the ImageNet ones, we stopped at 109 images labelled with their corresponding bounding box.

Which gives us **587 images** in the final dataset for the class "masked persons".

Data augmentation Finally, a good and simple way to get more data, and therefore create a more robust system is to perform some data augmentation. To do so, we added to the dataset an horizontal flip of each image, recomputing the new bounding box coordinates.

In fact, there exist many other transformations such as rotations, crops, scales and so on. However, you need *label-preserving* transformations and therefore prior knowledge about the invariant properties of the data against X or Y transformations.

To sum up, data augmentation permits to cover unexplored input space, improve the generalization capacities and prevent over-fitting. Nonetheless, the task is non-trivial except for horizontal flips from the necessity to conserve or convert your labels (bounding boxes), if necessary.

All in all, we implemented horizontal flips on each image and hence increased the size of the dataset to **1174 images** for the masked class, and consequently $\approx$ 2350 images in total with the additional helping class (non-masked person).

Testing set Our testing set contains images manually annotated from totally different source than the one of the training set, for obvious independence reasons. It adds up to $\approx$ 1100 images, with a lot of images without masked persons and just persons walking by. The reason behind this decision is inspired from the resulting low-precision in the first stage of the project. Indeed, a lot of non-masked persons where categorized as masked by the network and we wanted therefore to be sure that:

$$N \to \infty \implies Precision \to 0$$

Where N is the number of *non-masked persons* present in the testing set. Therefore, almost 90% of the testing-set ($\approx$990 images) are non-masked persons and the rest (10%) represents the masked category. This ensures that the testing set represents a good evaluation of the final objective, i.e. detecting masked persons on images without getting disrupted by the presence of non-masked persons.

iii. Web server and openHAB

In view of the initial idea: a real-time prototype fully working with our in-lab camera, we proposed a solution to capture the images from the camera steam, process them through the neural network and then trigger a light bulb in the case of an alarming detection.

To do so, we run a small script with "ffmpeg" to fetch the images from the camera stream and fill a folder with these in-frames. From there, we have a simple Python web-server processing the images with the help of the neural network, which outputs the bounding box of each relevant object found in the image, in another folder of out-frames. Finally, if any frame contains the aforementioned suspicious object, for the purpose of the demonstration, we light a bulb from an API connected to an "openHAB" server.

IV. Experiments and results analysis

The experiments and results of the different tuning performed are detailed in this section but if you do not want to skim through the whole text, the final decisions for each hyper-parameters is summed up at the end of their corresponding section.

i. Hyper-parameters investigations

On the following figures, the *top-left* plot represents the Average Precision of the "masked" class in blue and "non-masked" in orange (when present), *top-right* the recall of the system, *bottom-left* the precision of the system and *bottom-right* the F1-score, merging the previous precision and recall.

It is also important to realize that some of these tests were performed when only one class was being discriminated by the neural network, prior to the addition of "non-masked". The idea in these cases (with only the "masked" class), was to study tendencies and steadiness in the learning process more than the actual numbers, where with the two classes the point is more to comprehend the generalization capacities of the system.

Probability certitude threshold This threshold defines the minimal probability an object has to obtain to be considered in the bounding box, i.e. the minimal certitude the network has to claim before we confirm the detection of a "masked" or "non-masked" object:

$$P(masked|box) >= \eta$$

Where η is the threshold we are tweaking.

We investigated different threshold in Figure 9, Figure 10 and Figure 11.

From these figures, we can conclude that, despite the blatant fact that lower threshold makes the curves more unstable, this threshold has little to no impact on the overall performances. Indeed, we checked the probabilities given by the system and it confirmed the hypothesis that the threshold has minimal impact on the majority of cases since the network is either sure that there's nothing or something in the box, i.e. $P(masked|box) \approx 0$ or $P(masked|box) \approx 1$. The hypothesis is even stronger when we analysed the probabilities on errors, false negatives and false positives, they are either super high, ≥ 80, or super low, ≤ 10.

Therefore, since the main source of error (diminishing F1-score) is the low precision, while we always have a quite high recall, we decided to opt for a high value for this parameter to prevent unnecessary alarms trigger as much as possible and keep almost the same recall. Surely, the recall will be impacted, even to a lesser extent, but missing 1-2 frames (true positives) is acceptable for a system processing images at 100ms, if it is necessary in order to keep more than just a decent precision. Hence, we opted for the compromise of increasing the threshold.

Strategy kept: *80% threshold, namely* $P(masked|box) >= 0.8$ *before attesting the detection of a masked person*

Learning rate strategy Tweaking the learning rate policy is an important factor to achieve a flat minima and better generalize concepts in neural networks. In the style of the well-known "simulated annealing", we often favour exploration in the early stages of the learning process to later do exploitation, in the terminology of reinforcement learning.

The three different results on the learning rate decaying policy are illustrated on Figure 12, Figure 13 and Figure 14.

In spite of the good results of these techniques (with default "SGD optimizer" and standard momentum parameter), we decided to use the *RMSProp* optimizer and thus adopted its best learning rate policy:

- Policy: Inverse
- Base learning rate: 10^{-2}
- Power ρ: 0.75
- Gamma γ: 10^{-4}

$$current_{lr} = base_{lr} * (1 + \gamma * itern_{num})^{-\rho}$$

Strategy kept: *Inverse policy*

Gradient optimizer There exist many different gradient optimizers already implemented in deep learning frameworks, in caffe they are called "solvers" and they determine the network weights update rule:

- Stochastic Gradient Descent
- AdaDelta
- Adaptive Gradient

- Adam
- Nesterov's Accelerated Gradient
- RMSprop

All these solvers methods address the same optimization problem: loss minimization. To do so, they update the weights at time $t+1$ according to their value at time t, to the loss at time t and to some other specific techniques (first and second degree momentum and so on). Since the choice of the optimizer heavily depends on the application, each having their pros and cons for certain objectives, it is usually determined by empirical evidences. We analyse the performances of these optimizer on our problem in Figure 15, Figure 16 and Figure 17.

We further develop the intuition of the chosen optimizer (RMSProp), the exact formula is given by:

$$MS((W_t)_i) = \delta MS((W_{t-1})_i) + (1-\delta)(\nabla L(W_t))_i^2$$

$$(W_{t+1})_i = (W_t)_i - \alpha \frac{(\nabla L(W_t))_i}{\sqrt{MS((W_t)_i)}}$$

In more simple terms: we proceed by dividing the gradient by a running average of its recent magnitude. In this equation, α stands for the learning rate, determined by the policy of the previous paragraph "Learning rate strategy".

Strategy kept: *RMSProp with $\delta = 0.98$ rms decay*

Batch-size The batch size is a fundamental parameter in any machine learning algorithm exposed to gradient optimizer and back-propagation. Indeed, the idea is to compute the gradient on a sub-sample of the total input (2350 images in our case), for the sake of computational efficiency. However, for the gradient descent to properly converges towards a good minima (a flat one), the batch-size (*the sub-samble cardinality*) has to be finely tuned as exposed in [13]:

> "...and present numerical evidence that supports the view that **large-batch methods tend to converge to sharp minimizers** of the training and testing functions - and as is well known, sharp minima lead to poorer generalization. In contrast, **small-batch methods consistently converge to flat minimizers**..."

The four different experiments on batch-size levels are available on Figure 18, Figure 19, Figure 20 and Figure 21.

Strategy kept: *64 images per mini-batch, because of the stability of the learning process while keeping a high F1-score*

Regularization To prevent the network from over-fitting, it is customary to introduce some regularization factor to the weights. Indeed, complex models tend to over-fit and memorize the examples "by heart" instead of learning how to generalize concepts. There exist many techniques to govern the regularization of the neural net, such as drop-out, L_1 norm, L_2 norm, and many others. In our case, the backbone CNN (ResNet-50) already includes some kind of novel strategy to increase the network efficiency, while also playing the role of regularizer: *Batch Normalization* [14]. This innovative approach allows neural networks to be trained more efficiently, improves stability and permits to abandon older techniques of regularization such as drop-outs.

In our case, even if the data is quite small in comparison with usual datasets, the simple fact that our backbone CNN uses batch normalization prevented over-fitting altogether. On top of that, we added a small L_2 norm regularization factor to make the system even more robust.

Strategy kept: *$5 * 10^{-4}$ L_2 normalization factor on the weights*

ii. Examples

Here is a typical working example in Figure 22 and some of the errors that the system made in Figure 23, Figure 24, where the system respectively categorized a "non-masked" person as a masked one and then the system did not

detect the "masked" object in the frame. The high probabilities, even when it is wrong, come from the low number of examples that are presented to the network and therefore it tends to push the system to high probabilities since "nearest" examples from training frames are very few and sparse, hence the network cannot average over them. To alleviate the problem, as it will be discussed in the conclusion (section V), one should expose more examples to the neural network, each of different nature and from totally disparate scenes.

V. Conclusion and Future Work

During this project, we proposed a viable answer to the problem of object detection in the context of intrusion detection. Indeed, our solution processes images fast-enough with high recall and precision for competent authorities to be contacted in the case of an alarming detection. Moreover, since having a perfect balance between recall and precision (detecting "all" and "only" suspicious objects) is unrealistic for the moment, given current advances in machine learning, such a system should be used wisely and thus under supervision of a human before taking arbitrary actions.

We also have shown that obtaining stable results is a difficult task when images are from different sources, with varying quality, range of colors, dimensions and so on. Generalizing is even harder for the network in these cases, the number of examples one has to feed it with grows very fast as such cases arise. Hyper-parameter tuning is a crucial part of the stabilization of such a system and different techniques have been approached in this project, such as *decaying learning rate*, *different gradient optimizer*, *different batch sizes*, while many others exist that we did not have time to explore given the time-frame.

A key idea of this project was to correctly identify the reasons of the false-positive rate accrual as we increased the number of images in the testing set. Namely, the network had trouble distinguishing between masked and non-masked persons and as a result we chose to add a new class to the objectives of the neural network, i.e. detecting *masked*, *non-masked* and *other* (or background) objects.

As a result, it helped the system discriminate between the two and therefore tremendously improved the precision of the system while keeping the recall as high as before. On top of that, the system precision is also almost insensitive to the number of images with non-masked (or "normal") persons, which is one of the main requirement of such intrusion detectors.

Further improvements could obviously be obtained by gathering way more data to even better fine-tune the network, and mitigate the errors shown in subsection ii by helping the network generalize by, *sort of*, averaging examples. Additionally, since such security cameras often offer the possibility to gather infra-red stream (for night vision), one could perform information-augmentation through different sources of the same image. Consequently, by just adding the infra-red image as another channel, on top of the *RGB* one, the network could grab information from this one and further improve results. One could also install multiple cameras at different angles to corroborate their individual detections through a *master* system who would for example perform some kind of majority-voting (i.e. trigger an alarm iff the majority of the cameras detect a masked person). Thermal cameras could also be a major source of improvement by providing some unique information not represented by any previous channel (*RGB* or *Infra-Red*).

The code, help and dataset link can be found in the documentation located at the repository of the project: `https://c4science.ch/source/ipg-semester-project/`

References

[1] Z. Xiao, X. Lu, J. Yan, L. Wu, and L. Ren, "Automatic detection of concealed pistols using passive millimeter wave imaging," in *2015 IEEE International Conference on Imaging Systems and Techniques (IST)*, pp. 1–4, Sept 2015.

[2] R. Gesick, C. Saritac, and C.-C. Hung, "Automatic image analysis process for the detection of concealed weapons," in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09, (New York, NY, USA), pp. 20:1–20:4, ACM, 2009.

[3] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016.

[4] R. Girshick, "Fast R-CNN," in *International Conference on Computer Vision (ICCV)*, 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[6] R. Olmos, S. Tabik, and F. Herrera, "Automatic handgun detection alarm in videos using deep learning," *CoRR*, vol. abs/1702.05147, 2017.

[7] S. Ray, S. Das, and A. Sen, "An intelligent vision system for monitoring security and surveillance of atm," in *2015 Annual IEEE India Conference (INDICON)*, pp. 1–5, Dec 2015.

[8] J. H. Hosang, R. Benenson, P. Dollár, and B. Schiele, "What makes for effective detection proposals?," *CoRR*, vol. abs/1502.05082, 2015.

[9] J. H. Hosang, R. Benenson, and B. Schiele, "How good are detection proposals, really?," *CoRR*, vol. abs/1406.6962, 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[11] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," *arXiv preprint arXiv:1605.06409*, 2016.

[12] T. Vu, A. Osokin, and I. Laptev, "Context-aware CNNs for person head detection," in *International Conference on Computer Vision (ICCV)*, 2015.

[13] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016.

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

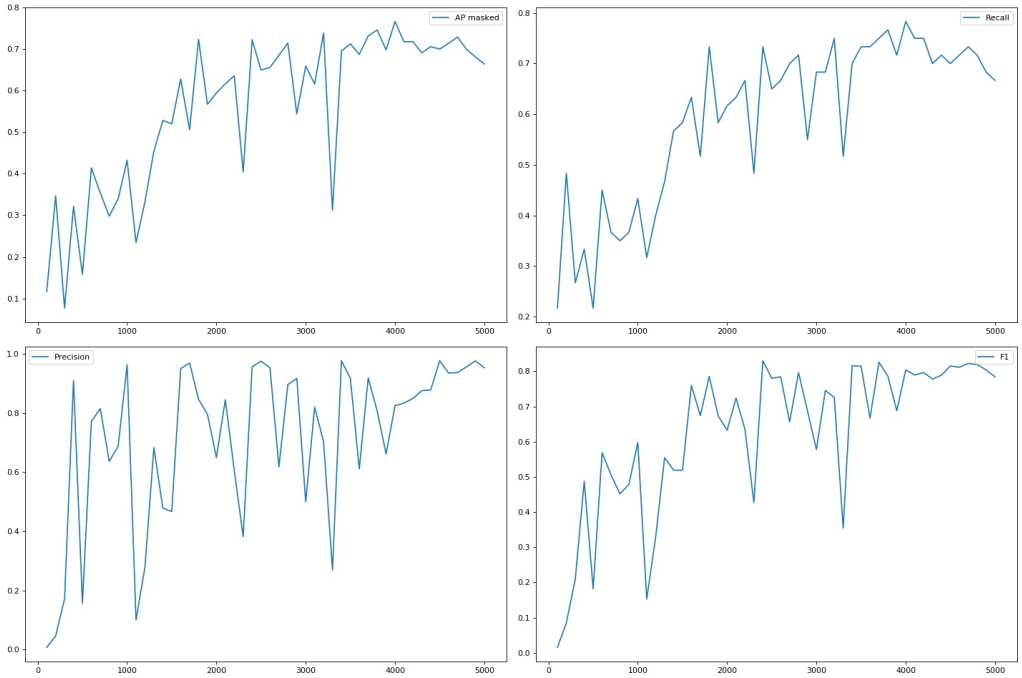

Figure 9: *Threshold of 20% before validating an object in the bounding box*

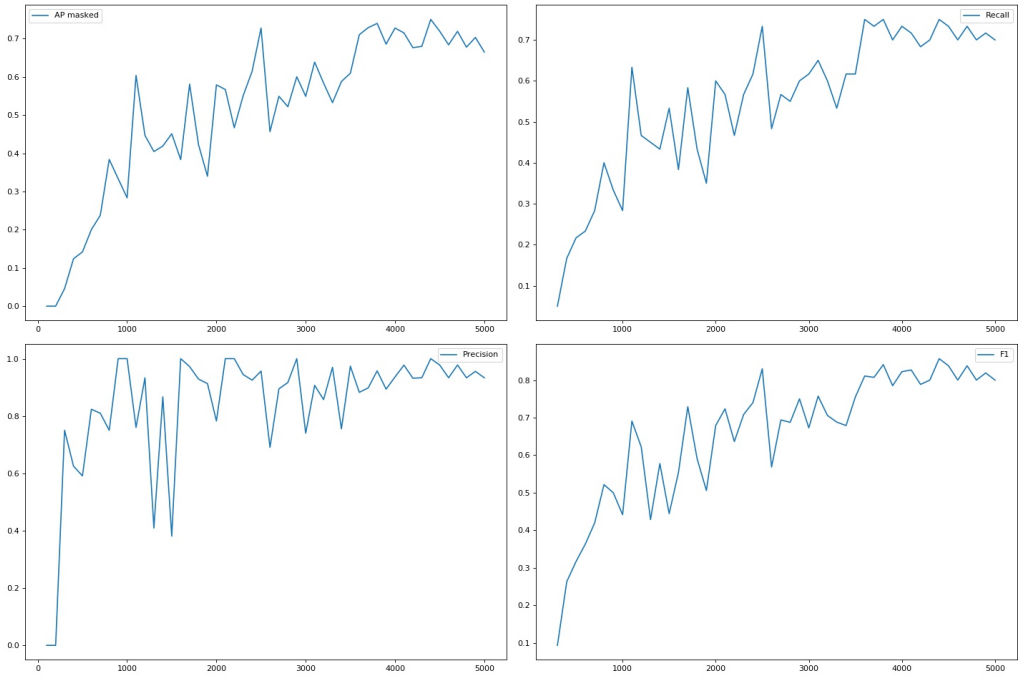

Figure 10: *Threshold of 50% before validating an object in the bounding box*

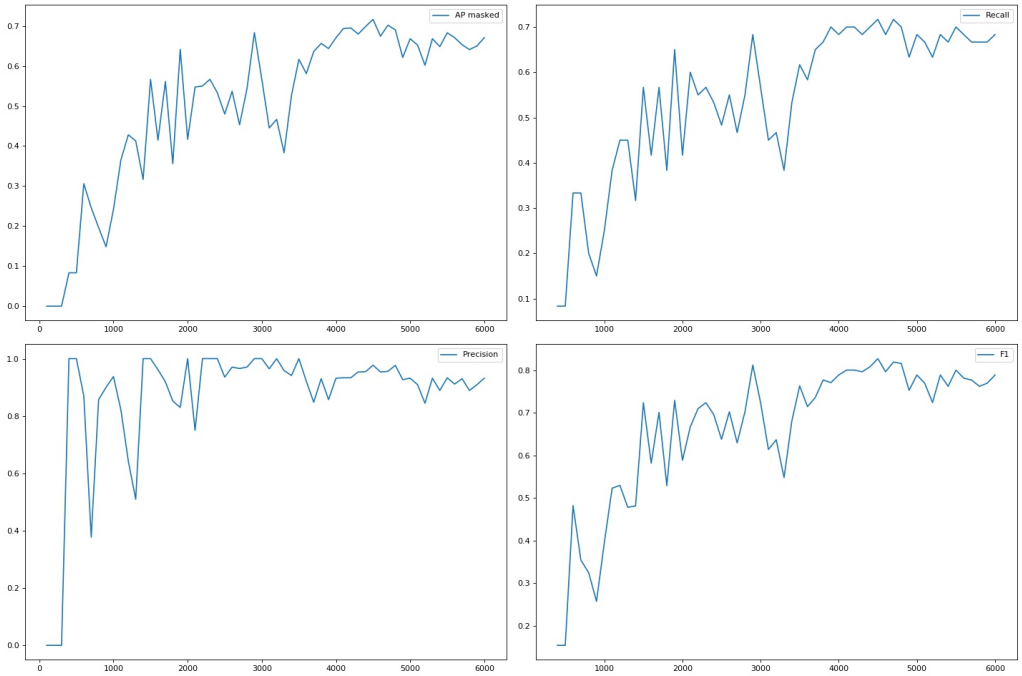

Figure 11: *Threshold of 65% before validating an object in the bounding box*

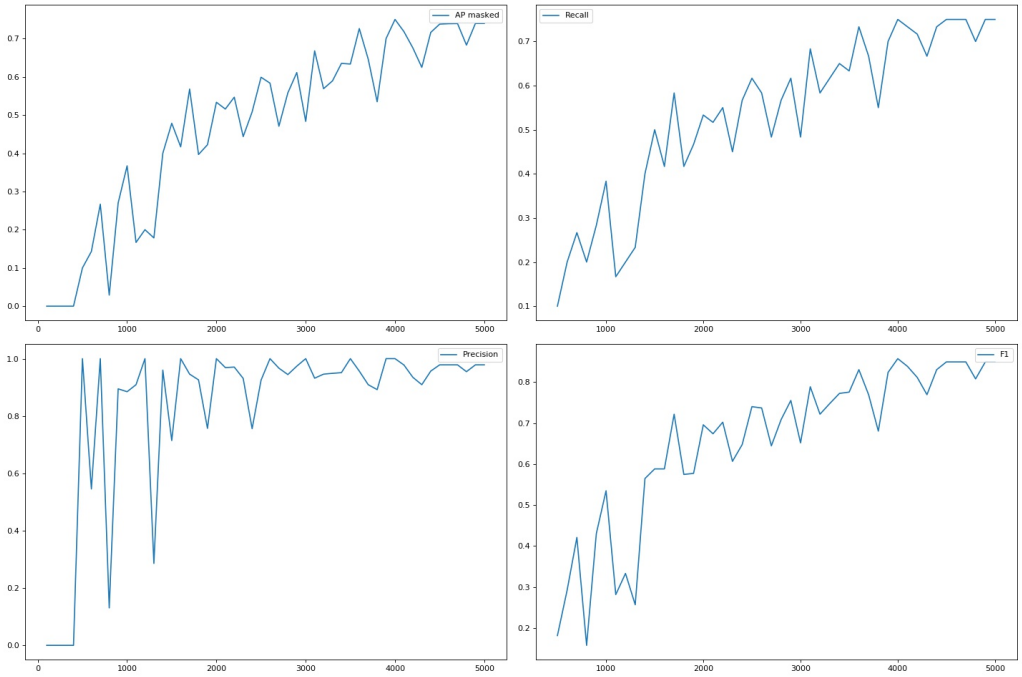

Figure 12: *Dividing the learning rate by 10 at 4000 iterations out of 5000 in total*

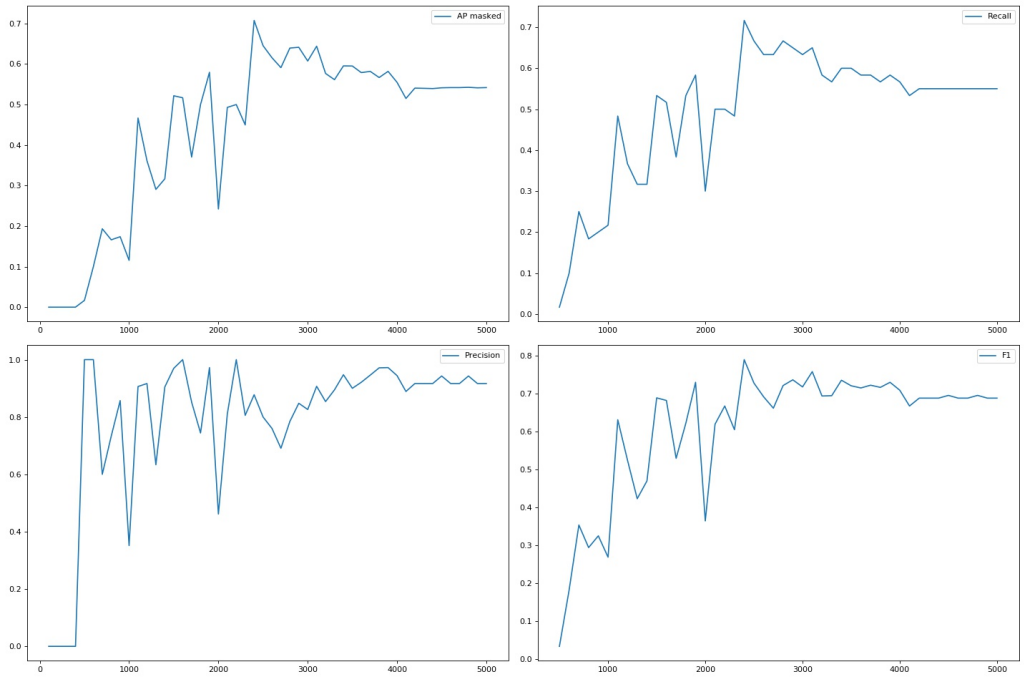

Figure 13: *Dividing the learning rate by 10 at 2500 iterations and then again by 10 at 4000, out of 5000 iterations in total*

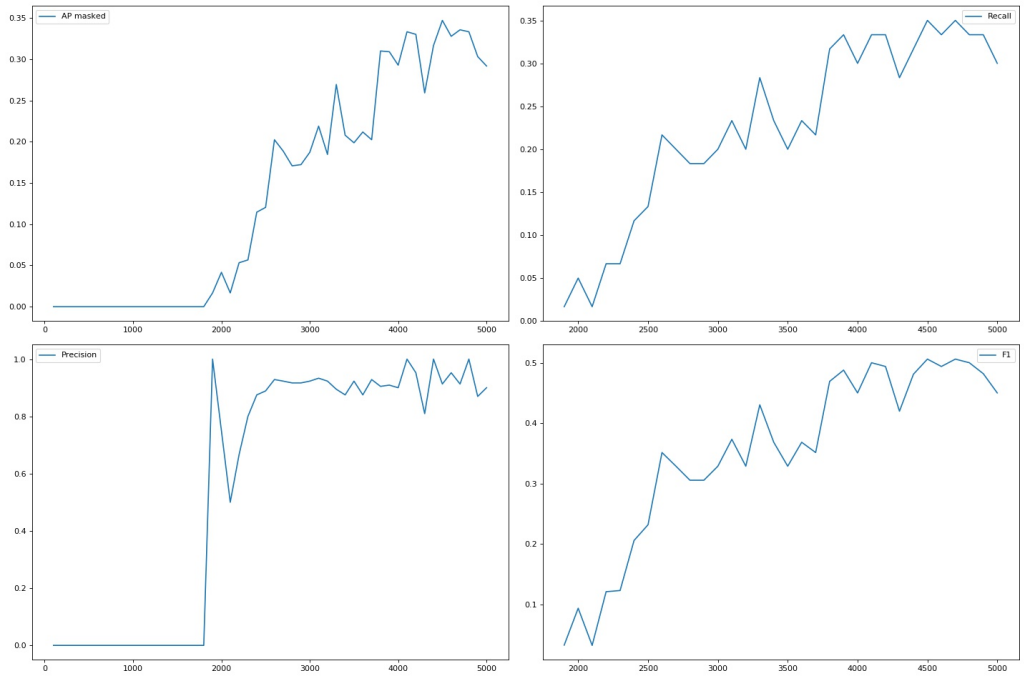

Figure 14: *Lower learning rate since the very beginning and no decay during the 5000 iterations*

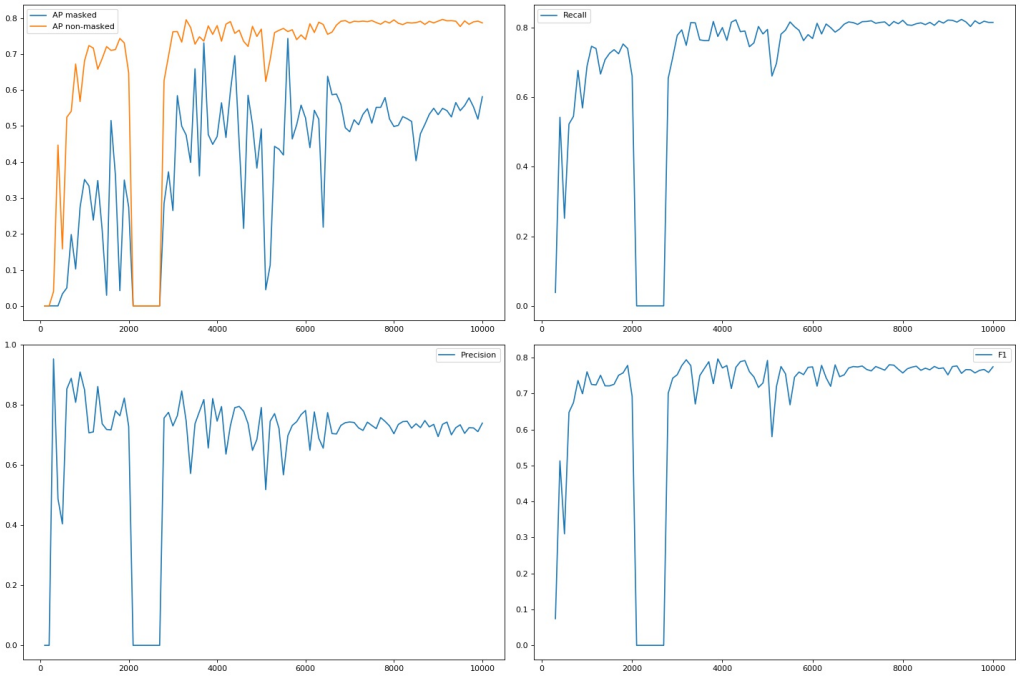

Figure 15: *Default optimizer "Stochastic gradient descent", with inverse learning rate policy*

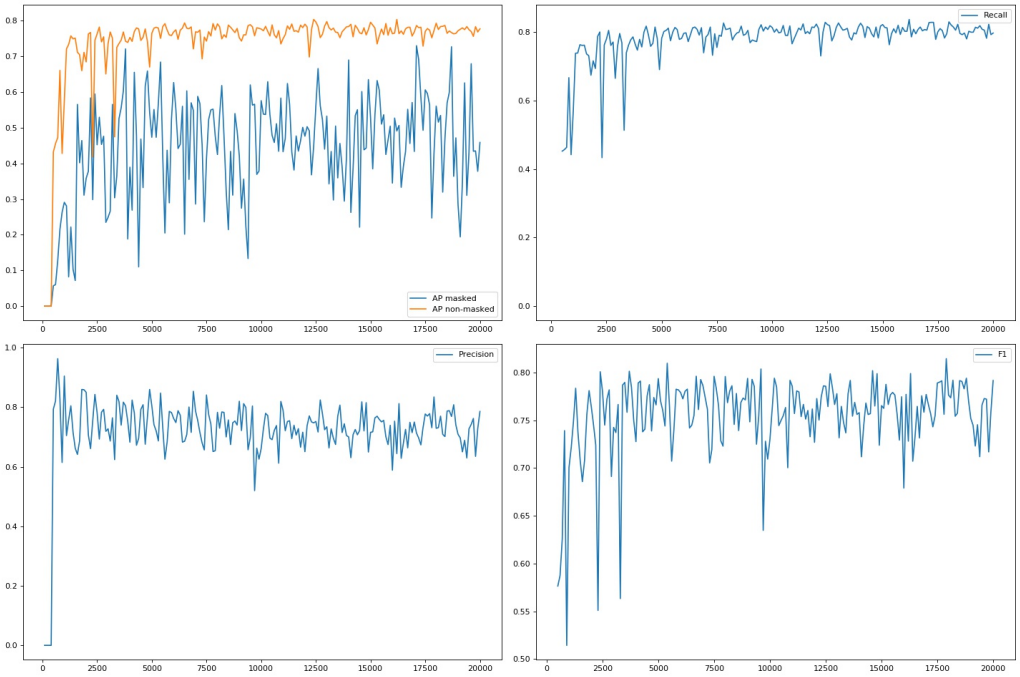

Figure 16: *Adam optimizer, very unstable and therefore apparently not suited for our application*

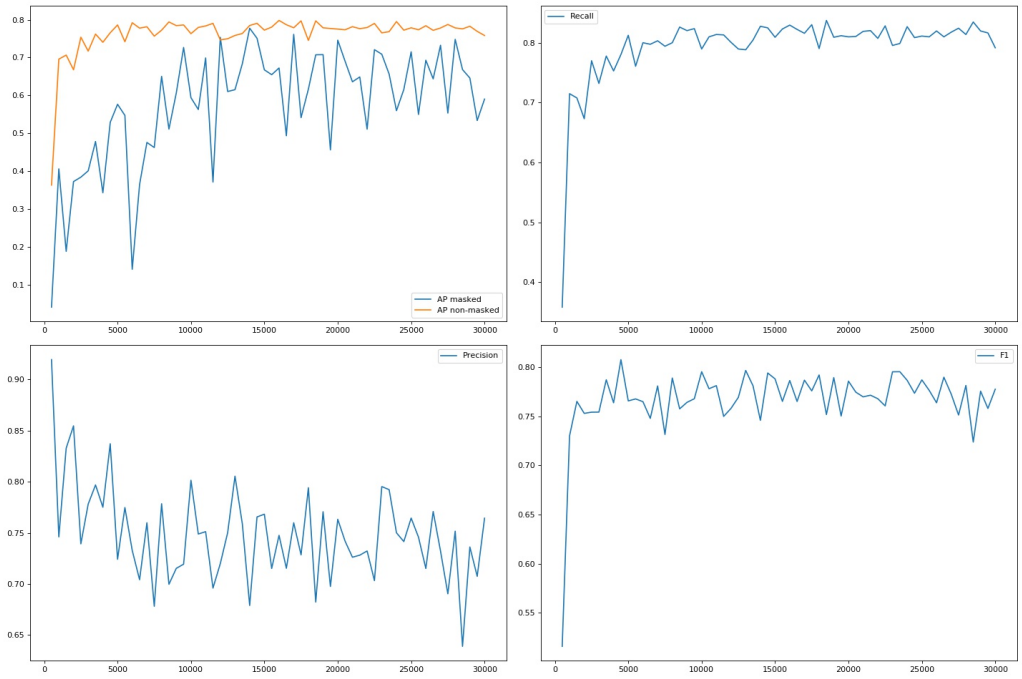

Figure 17: *RMSProp optimizer which offers a very good F1-score, for a recall and precision level almost equal, oscillating between 75%-80%*

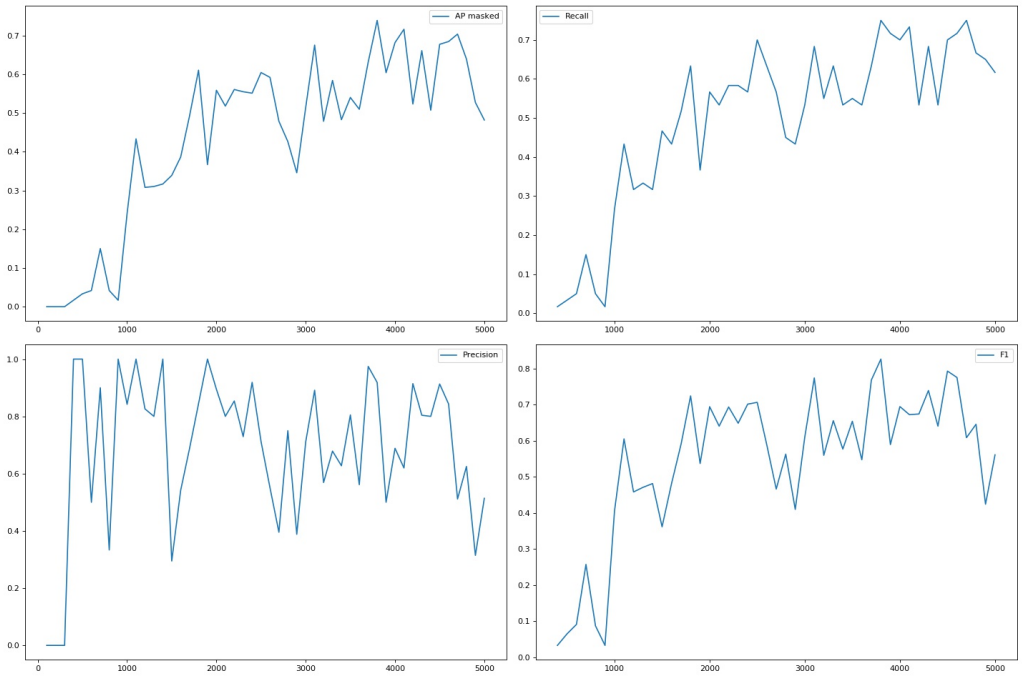

Figure 18: *32 images per mini-batch*

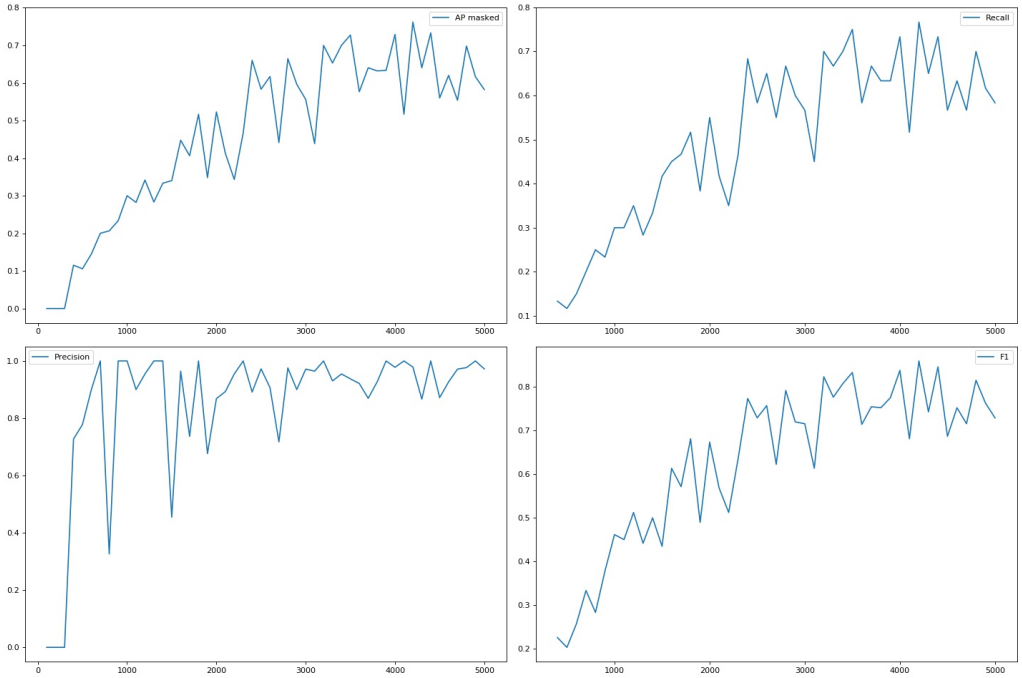

Figure 19: *64 images per mini-batch*

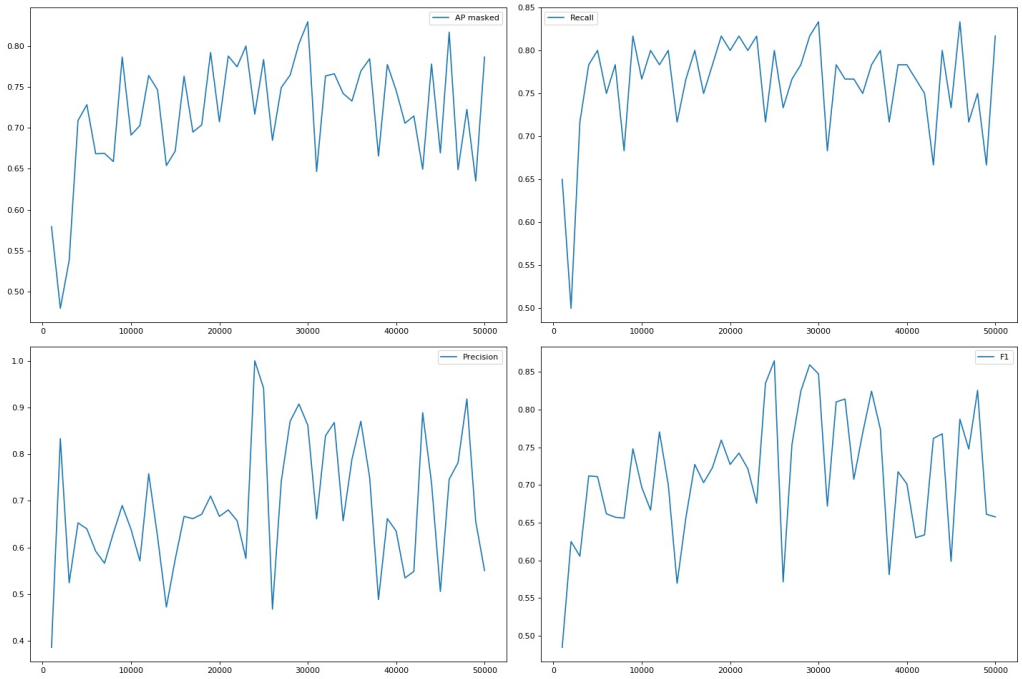

Figure 20: *128 images per mini-batch*

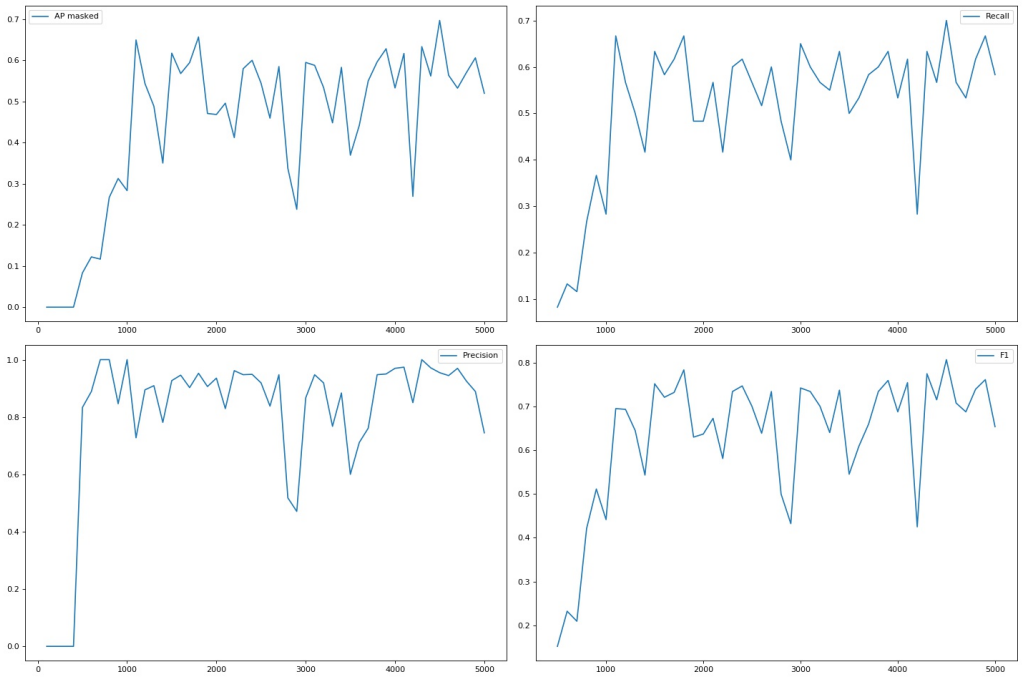

Figure 21: *256 images per mini-batch*

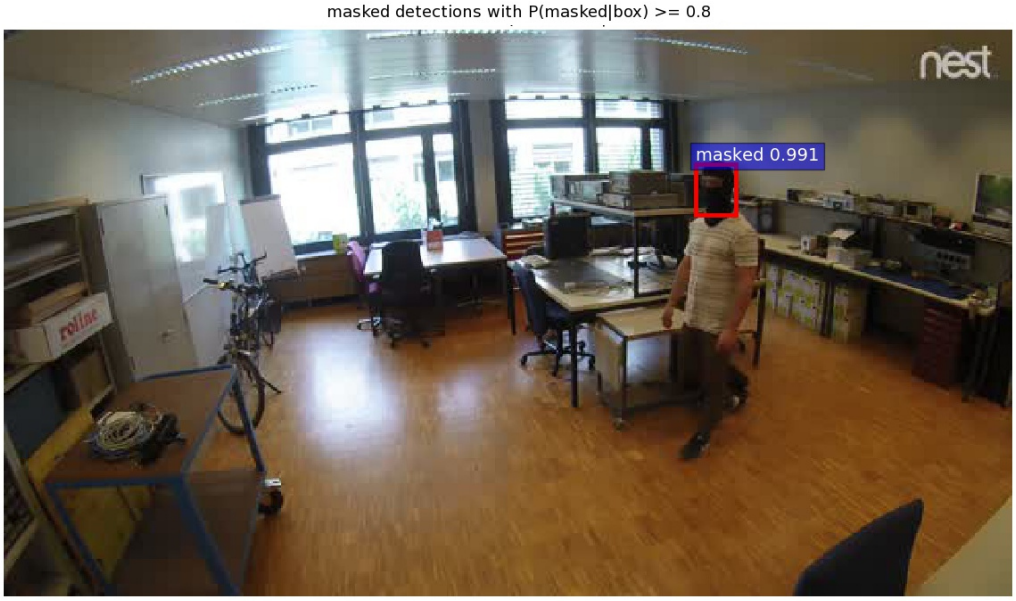

Figure 22: *A true positive example, where the system correctly and with high certitude detected the* masked *object*

Figure 23: *Typical example of the false positives, the certitude probability of the network is so high (99.4%) that the high threshold has no impact in reducing these errors. Note that the image coloured in black and white fools the system even more*

Figure 24: *Example of a false negative, the mask is not detected even if it seems clear to a human eye*