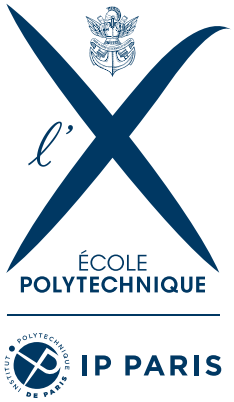
Eulalie Boucher
Maël Marguet
Cédric Javault



December 2020

MASK AND SOCIAL DISTANCING DETECTION IN VIDEOS

INF634 Computer Vision

Contents

1 Introduction

In December 2019, a new disease has emerged from Wuhan, China. A few weeks later, this virus had spread all around the world, and changed for sure the world we knew. We had entered the COVID-19 era. But to what extend has our world changed? One can just have a look at pedestrians on the streets to see the first change. Everyone is wearing a mask, or at least everyone should. A second change, perhaps less perceptible, is social distancing. Big groups are forbidden, and people are asked to stay away from each other as much as possible. A minimal social distance has been established to limit the spread of the virus. All these changes have motivated our subject: detecting people, detecting whether they are wearing a mask or not and being able to tell if they respect social distancing.

For this computer vision project, we take a video as an input, cut it into frames (images), and then:

1. Detect people and assign bounding boxes.
2. Assess whether a mask is worn or not and assign a bounding box to the faces.
3. Combine information from those two detectors to propagate the mask/no mask label from faces to people.
4. Track the people across the different frames of the video
5. Estimate a dense depth in the images, compute 3D positions of the detected people, and estimate the inter-person distance.
6. Finally, output the main results in a new video so that it is easy to understand what the computer computed.

2 Object detection with YOLO

Object detection is a computer vision technology that deals with detecting instances of objects of a certain class or category in digital images and videos. The concept behind this is that every object class or category has its own features that helps, us humans, in understanding the class. A simple example is when looking at a building, and detecting windows. Windows are rectangles, therefore we look for perpendicular corners and equal sides. Object detection models makes use of these features while training, in order to find instances of various objects at test time. A very specific application of object detection is people and face detection, where specific features are used to locate eyes, nose and lips for example.

Many methods exist for object detection. These generally fall into either machine learning-based approaches of deep learning-based approaches. For this project, we concentrate on deep-learning methods, and especially on YOLO "You Only Look Once".

2.1 Overview of YOLO "You Only Look Once"

Most of this section is a reference to the article "You only look once: Unified, real-time object detection", Redmon et al. [2015].

YOLO, standing for You Only Look Once, is an object detector that, to the contrary of most object detectors, re frames object detection as a regression problem, going straight from pixels in the image to bounding box coordinates and class probabilities. Indeed, other networks perform detection on various regions of the image and thus ends up performing prediction multiple times for various regions in an image. This is not the case for YOLO, which acts more like a fully convolutional neural network as it passes an image once through the network, and outputs the prediction. Therefore, YOLO uses features from the entire image to predict each bounding box, and predicts all bounding boxes across all classes simultaneously.

Figure 1 shows an overview of the way the YOLO model functions. In general terms, YOLO divides the input image into an $S \times S$ grid. If the center of an object falls into a grid cell, then this cell is responsible for the detection of that object.

Each grid cell predicts B bounding boxes and confidence scores for those boxes. Each bounding box consists of 5 predictions: x, y the coordinates of the center of the box, w, h the width and height of the bounding box and finally, the confidence prediction which represents the IOU score between the predicted and any ground truth box. These confidence scores reflect how confident the model is that the box contains and object, and also how accurate it thinks the box is. Formally, this is defined as $P(\text{object}) \times \text{IOU}_{pred}^{truth}$. Each grid cell also predicts C conditional class probabilities $P(\text{Class}_i|\text{Object})$.

At test time, we multiply these conditional class probabilities and the individual box confidence predictions: $P(\text{Class}_i|\text{Object}) \times P(\text{Object}) \times \text{IOU}_{pred}^{truth} = P(\text{Class}_i) \times \text{IOU}_{pred}^{truth}$, which gives us class-specific scores for each box. Therefore, these scores encode both the probability of that class appearing in the box, and how well the predicted box fits the object.

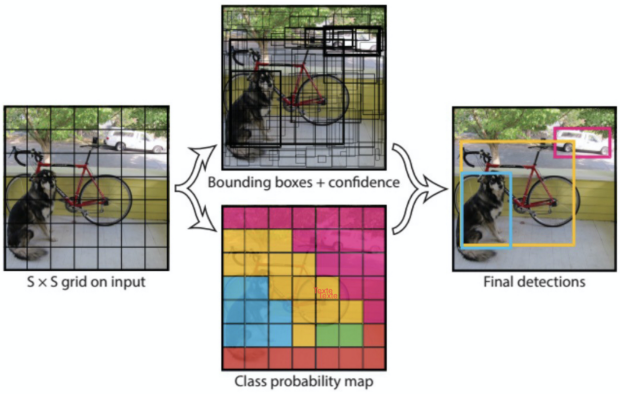


Figure 1: Overview of the YOLO, You Only Look Once model

2.2 Specifications of YOLO, You Only Look Once

Inspired by the architecture of the GoogLeNet model for image classification, the YOLO architecture has 24 convolutional layers followed by 2 fully connected layers and uses 1×1 reduction layers followed by 3×3 convolutional layers to replace the inception modules used by GoogLeNet. Figure 2 depicts this architecture.

The optimization loss is based on sum-squared error. This choice is motivated by the fact that the problem is treated as a regression problem and that it is relatively easy to optimize. However, naturally it does not perfectly align with the goal, which is to maximize average precision. It accords the same weight to both localization errors and classification errors, which is not ideal. Also, in most images, there are a lot of grid cells that do not contain any object, pushing the confidence scores of those cells towards zero, and thus overpowering the gradient from cells that do contain objects.

To counter this, the model uses a modified version of the sum-squared error, in which we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that do not contain objects. We use two parameters λ_{coord} and λ_{noobj} to accomplish this.

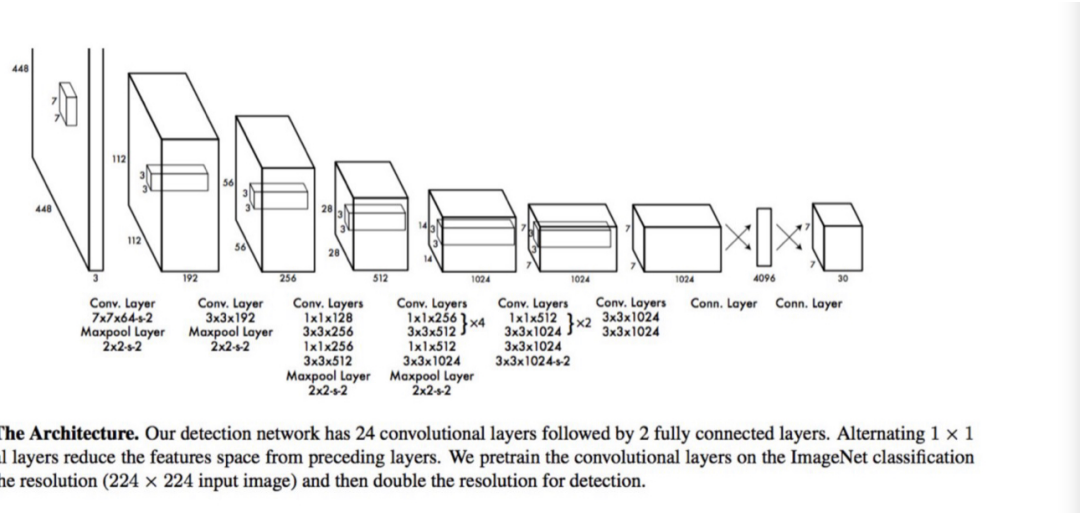


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Figure 2: The Architecture

2.3 Advantages, Limitations and Versions

The YOLO model has several benefits over traditional methods of object detection. First, YOLO is extremely fast, since detection is seen as a regression problem, there is no need for a complex pipeline. The network runs at 45 frames per second and can be used for real-time video streaming with less that 25 milliseconds of latency. Secondly, YOLO reasons globally about an image. This is an advantage compared to Fast R-CNN for example, which mistakes background patches for objects because it can't see it in the larger context.

However, YOLO also has a few limitations. YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that the model can predict. Therefore, it struggles with small objects that appear in groups (like birds for example). Also, due to the fact that the model looks at an image as a whole and has multiple down sampling layers from the input image, the features used for prediction bounding boxes can be relatively coarse.

These limitations led to 4 other versions of the initial YOLO, all getting higher accuracy rates and becoming better at detecting all kind of objects. For our project we will be using the latest (fifth) version, YOLOv5.

3 Tracking through frames with SORT

This section presents the principle of tracking and the tracker we decided to use. It is mainly a reference to this article Alex Bewley and Upcroft [2017].

Tracking consists in identifying the same objects across subsequent frames of a video. Several methods exist to achieve this. First, the tracking can be online or batch. Batch tracking consists in tracking an object according to batch frames, meaning that the tracking is made batch to batch. The advantage of this method is that when the system is looking at objects in frames, it can look at other frames, which are past or future frames, as long as they are in the batch. It won't be the case here since we choose an online tracker, meaning that the objects are identified only due to the past frames.

Tracking relies on object detection: tracking and detection are two coupled-tasks. There are two ways to use trackers, as depicted in the figure 3. Here, MOT stands for "Multiple Objects Tracking".

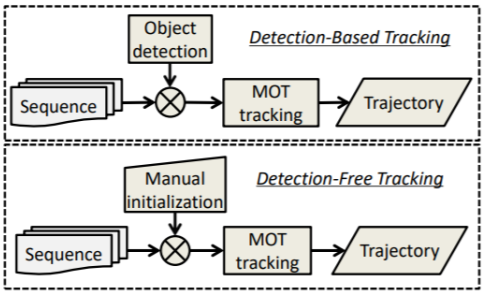


Figure 3: Two different ways to use trackers: DBT vs DFT

A common use is Detection-Free Tracking, which means that at first a user specifies the zone to track. In this case, the initialization of the tracker is manual. A second approach, more generalizable, is to feed tracking with previous detection, meaning the whole system is composed of a detection system, and a tracking system that tracks detected objects. The goal of Detection-Based Tracking is to associate, at each frame, the detected objects and the corresponding trackers. It is this approach we choose.

3.1 The SORT Tracker

The tracker we consider is called SORT tracker, for Simple Online and Real-time tracker. Why are we considering this one ? The answer is depicted in the Figure 4.

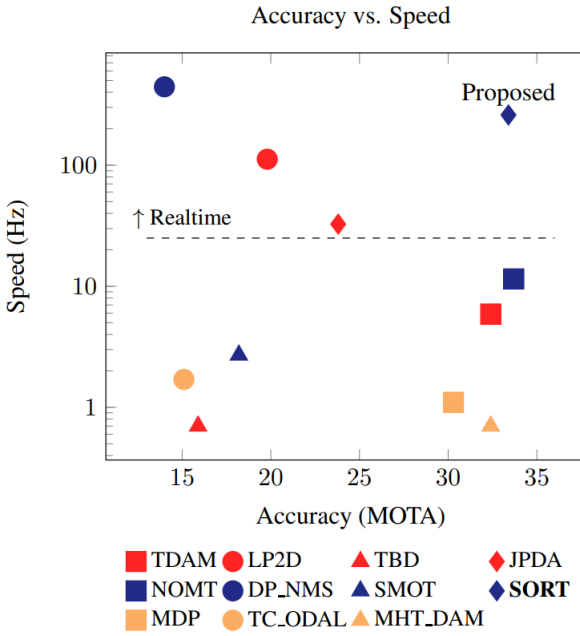


Figure 4: Benchmarks of several trackers

On this figure, one can see two characteristics of several trackers: speed and accuracy. We can see that the most accurate trackers, such as the NOMT or TDAM are also the slowest. On the contrary, we can also see that the fastest trackers, like DP_NMS, are the less accurate one's. In the top right corner of this figure stands the SORT tracker. The SORT tracker can be considered as the best trade-off between speed and accuracy.

How does it reach such performances ? It uses a really simple system that proves to be efficient in most cases. The trackers, in a video, are only seen as bounding boxes. Only the position, the size and the velocity of the bounding boxes are considered: the features of the detected objects are not taken into account.

3.2 Model

At each frame, one has (i) detected objects and (ii) targets (trackers previously detected); the goal is to associate targets and detected objects. As long as the tracker has not been associated to a detected object, it is considered as a target. A target state is defined as follows

$$x = [u, v, s, r, \frac{du}{dt}, \frac{dv}{dt}, \frac{ds}{dt}]^T$$

where u and v are the positions of the pixel at the center of the target bounding box, s and r are respectively the scale and the ratio of the target bounding box. When a detected object is associated to a target (we will see how the association is processed later), the target is updated thanks to the new detected object bounded box. The velocity components are updated optimally. If a target does not find a detected object match, its component are updated thanks to a simple linear velocity model.

3.3 Data association

First, the new location of the target bounding box is predicted according to its model. Then, the assignment between targets and detections is made thanks to the IoU metric. A minimal IOU_{min} is required for an association to happen. This specific threshold permits to handle short occlusion. The tracker corresponding to the occluded object is, during the occlusion, not assigned and is re-assigned when the object appears again.

3.4 Creation and Deletion of the trackers

A new tracker is created when the maximel IoU of a detection is less than IOU_{min}, signifying the existence of an object untracked so far. A tracker is then initialized with the following state: bounding box parameters of the detected object and the velocity set to 0. The deletion framework is even simpler: a tracker is deleted if it has no detection association for at least T_{lost} frames.

The simplicity of this tracking method makes it an ideal one for our framework. Indeed, due to its simple model, this tracker only behaves badly if the detected objects have random or fast changing movements. This is not our case at all: we only consider humans walking, i.e. a relatively slow and continuous movement. In addition, this tracker is ideal because it does not depend on the point of view: the camera does not need to be stationary (ours and not).

4 Social Distances

Estimating distances between two people in the 3D world, out of a simple image, is challenging. Essentially, a robust dense estimation of the depth in the field of the image is needed (dense: one depth per pixel). Then, with the knowledge of the camera intrinsic parameters[1] (focal length, position of the optical center, and skew coefficient), simple computation leads to the 3D coordinates, which themselves give the person to person distance through Euclidean distance.

"Old" non AI-based methods were developed to address the depth estimation issue, but we decided to use a AI-based framework, and namely the one proposed by Godard et al. [2018] that uses self supervised learning to train a network predict a dense monocular depth estimation. The authors claim they achieve state of the art

[1]We do not need the 3D coordinates in the real word but only 3D coordinates to compute the Euclidean distance between two points. Therefore, the extrinsic parameters (rotation and translation) can be set to Id and null.

performances. We did not dig too much into the paper, but rather directly adapted the code provided by the authors.[2].

5 Implementation details

We implemented two different YOLO. The first one was to detect people in an image: we used a YOLOv5 found on Github and that had already been trained on the 80 classes of COCO. We checked the results on different images in various contexts and the results are very good with high confidence for the people class that we were targeting.

We did not find such an already implemented tool for the face/mask detection and decided to build it. To do so, we trained another YOLOv5 with a first dataset found on the web but the results were poor. We then changed for a more robust dataset taken from Kaggle [3] and got much better results. This dataset consists of 853 images displaying many faces (3232 with a properly worn mask, 717 without mask and 123 with a mask that was not properly worn). To ease the training and because we were concerned that 123 would not be enough, we grouped the 3232 and the 123 in a unique 'Mask' class and of course left the 717 in a 'No Mask' class. This time, after 40 minutes of training, the results were impressive. An example is shown on Figure 5.

Figure 5: Example of output of our YOLOs. Left: people detection. Right: Mask detection.

Then, we implemented a 'Merge YOLO' algorithm which works as follows in each image. An illustration is shown on Figure 6: for each detected person, it looks at all the bounding boxes of the faces intersecting the bounding box of the people, computes a score based on $IoU \times Confidence$, keeps the best face and the following outputs:

- A class: Mask, No mask, Unknown (if no face was found)
- The bounding box of the people
- The bounding box of the face (or again the one of the people for the 'Unknown' class.

Then, we implemented the SORT tracker. We started from a baseline found on Github to save time. We set $T_{lost} = 0.7sec$ (more precisely to 20 frames) as this allows an occluded person to reappear after another person in the foreground crossed. The result was very good as the tracker successfully follows people across the frames. However, we had to make two changes:

1. The code we found was perfect to display the images and to build a video but we had to create another output (a text file containing the final bounding box coordinates) to serve as an input for the social distancing algorithm.

[2]Code available on Git: https://github.com/nianticlabs/monodepth2
[3]https://www.kaggle.com/andrewmvd/face-mask-detection

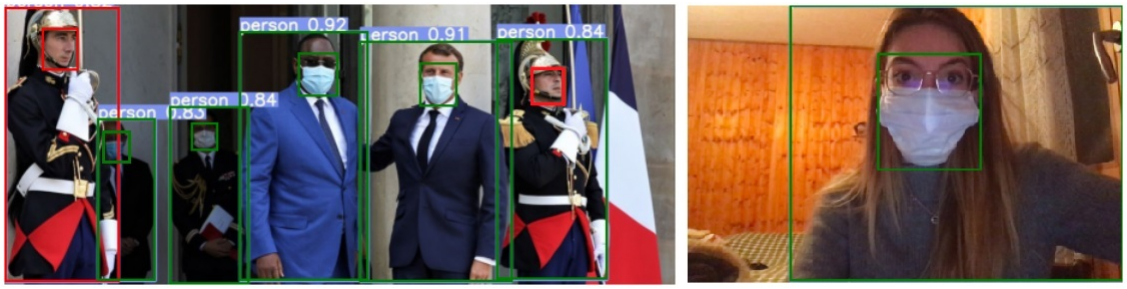


Figure 6: Output after the Merge Yolo. Here, all the faces are detected and everyone is either red (no mask) or green (mask).

2. In practice, when a person is first detected, he is far from the scene and the algorithm is likely to output the 'unknown' class for the face. However, when the same person moves and get closer, it will be easier for the algorithm to know whether or not he/she is wearing a mask. Originally, the tracker takes the class of the first appearance of an object and keeps this all along. To remedy this, we incorporate a variable into the tracker which tracks the class predicted by the merged YOLO at each frame and use this to back-propagate the majoring class to all frames in which an object appears (if the same person is seen in images with a mask and without a mask, nothing is done; changes of class are only applied for the 'Unknown' class). See Figure 7 for an illustration.

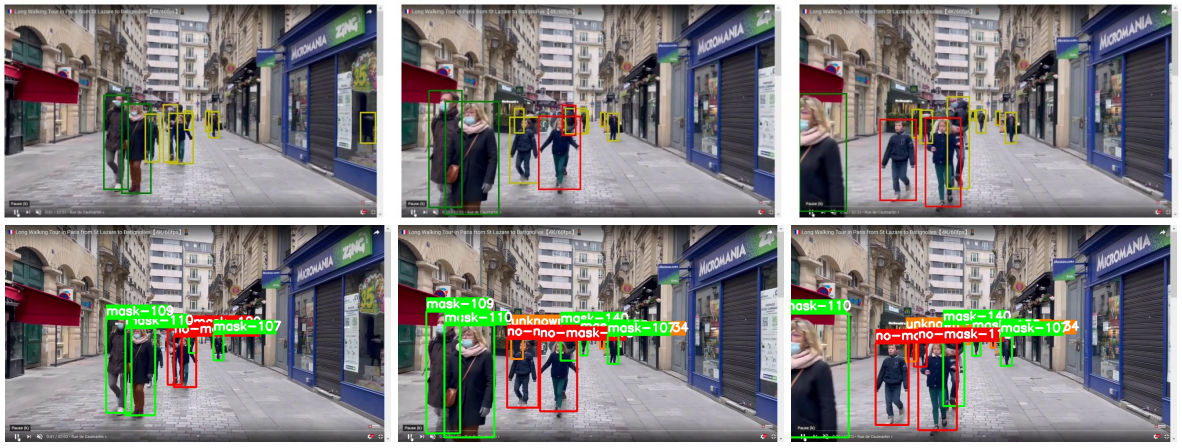


Figure 7: Frames 860, 886 and 898 of our second video. Above: result of the 'Merge YOLO' before the tracking. Below: same after the tracking: ID are propagated and the children are always from no 'No mask' class (ie red bounding boxes)

For the social distancing issue, as explained earlier, we started from the code provided by Godard et al. [2018]. Then, our work included:

- A proper understanding of the output (NPY file for the depth map + JPEG for visualization): as the code was almost not commented, we had to perform extensive checks to be sure that we properly understood the results. For example, the values in the NPY file are not the distance to the camera but its inverse, and an unknown scale factor is included. To tune it, we computed the average speed of the pedestrians and set the scale factor so that this average speed was 4.5 kilometers per hour.
- The estimation of the camera parameters: the position of the optical center can be deduced from the intersection of the vanishing lines (it was not at the center of the image as can be seen in Figure 8); we

set the focal length $f_x = f_y$ from the width of the street (estimated at 5 meters) and the skew coefficient to 0.

- Even in a quite easy environment like a street, the dense depth estimation is not very precise as illustrated on Figure 9: the distance to the camera was better estimated from the center of the bounding box of the person than from the one of its face. Additionally, the predictions of the position are smoothed by averaging over 20 frames.

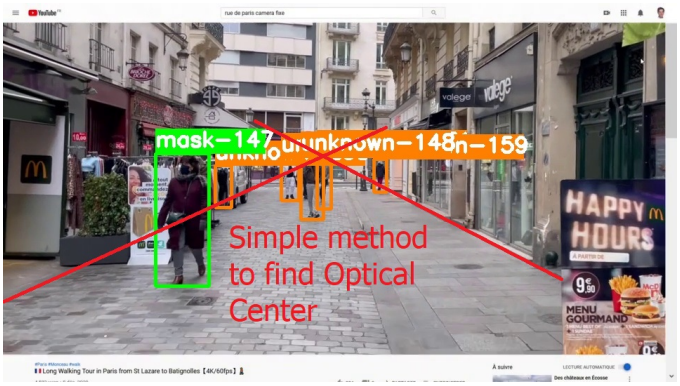


Figure 8: Determination of the optical center with the vanishing lines

Figure 9: Lack of precision of the Depth estimation. Look at the man at the bottom right. The depth estimation will be much more precise from the bounding box of the person, because the dense depth estimator is noisy and misleading in the region of the head.

We can compute the 3D positions for each tracked person and all person to person distances. However, we choose a few people and only displayed their positions and inter person distances so that a human mind can reasonably deal with the information.

We finalised our code using basic functions from OpenCV to cut videos into its different frames, tag each frame with bounding boxes, add some text and build videos out of these resulting images.

6 Results

We mainly worked on two small videos that we found on the web. While the second one includes detections, tracking and social-distancing estimates, the first one is missing this last element.

6.1 Short video of President Sarkozy (10 sec)

The first video we have been working on is a short sequence of Nicolas Sarkozy walking in a closed room, surrounded by colleagues. This scene also contains immobile safety guards. It is an interesting video because we can see the main protagonist walking in front of the camera and then to the left, therefore with another point of view, where we see the people from their side profile.

The results are quite good. Every person in the video is detected and tracked: this means that our people and mask detector together with our tracker are properly doing their job. Although we have no proper metric to evaluate our system, visually, we are really satisfied by the results. For example, Nicolas Sarkozy and his colleague on his left appear on all the frames, and are properly tracked along the video (same Id number). We only lose one person because of a long occlusion as shown on Figure 10.

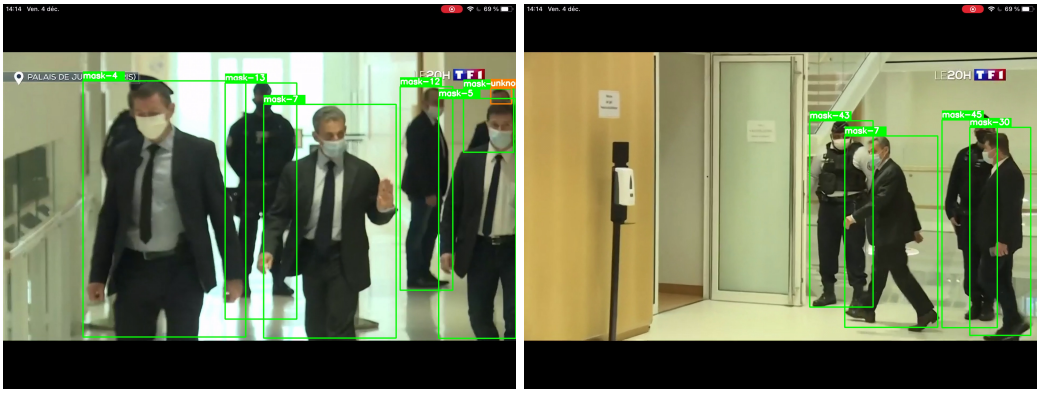


Figure 10: N.Sarkozy “Mask-7” and his colleague on his left “Mask-unvisible” becomes “Mask-30” (he disappears during 2 seconds) are detected during the whole sequence

Nevertheless, we can make several critics about ours results.

- As one can seen on Figure 11, a tie is detected. This is due to a mis-classification from the YOLO responsible of detecting people which is propagate to the Merge YOLO and then to the tracker. To overcome such problem, we could add a minimal number of frames required for a tracker to appear. This idea is better than adding a minimal size of the bounding box, because a person can be detected from far away, thus having a small bounding box.
- Another interesting problem can be seen comparing Figures 10 and 11: “Mask-4” becomes “No-Mask-24”. Actually, there are two problems: (i) Because the object detector suddenly outputs a much smaller bounding box, the tracker looses the person and initialize another one (we could tune the Min_{IoU} to

overcome this problem) and (ii) For this second tracker, the mask detection is not easy because the person is from profile: his face suffers from a lot of miss-classifications, and since the main label kept after post-processing is the majoring one, he is classified as "No-Mask" all along.

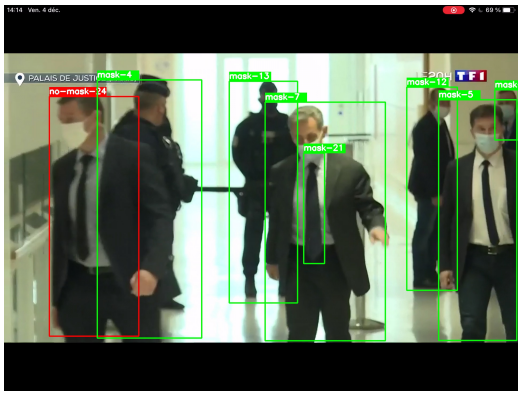


Figure 11: Tie miss detection (middle) and wrong detection ("No-mask-24")

Interestingly, the depth estimator could not give us consistent results and therefore we had to forget about social distancing for this first video. Figure 12 illustrate how completely the depth map changes in just a few frames and is therefore totally misleading. The main explanation is probably the absence of vanishing lines in those indoor images (the French windows illuminating at the beginning of the video or the plexiglass structure in the second part probably does not help!). Even if the depth estimation would had been successful, we would have struggled with the camera parameters as a defocalization takes place along the walk...

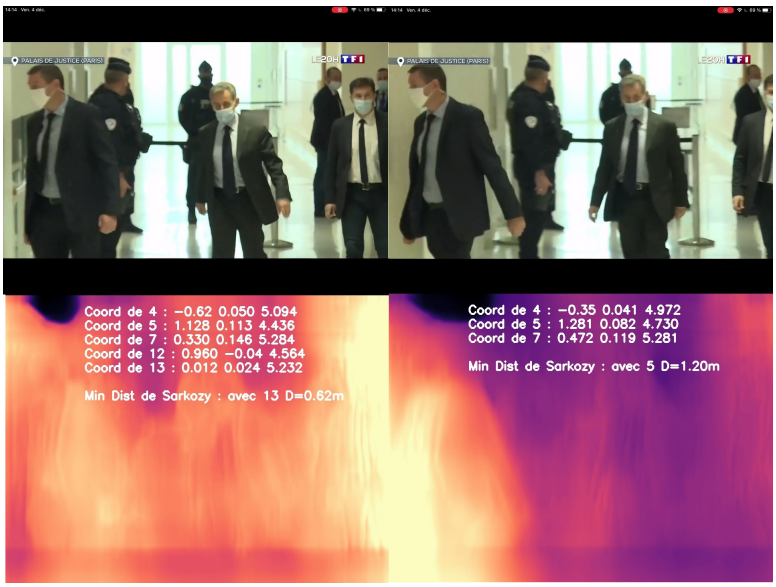


Figure 12: Frames 180 (left) and 193 (right) are very close but their depth maps are really different: at least one of them is false!

6.2 Video in the street of Paris (45 sec)

We found the second video on Youtube, acquired it, and used ShotCut to process it (cut the beginning and the end, remove the audio track, downsize it a lower resolution of 960 × 540). In terms of computation time, each

YOLO detector needs 0.27sec per frame leading to a total of 12 minutes for the 1332 frames. The merging procedure then takes again a few minutes (mainly because we wanted to save the images for debugging purposes). Additional time is needed for the tracker, for the Depth Estimator, and to fuse the images for the final output. The entire process takes about our hour to generate the final video.

The results are even more satisfying and close to what we expected then for the first video. Both the detection and the tracking work very well! As previously explained, we have a post-processing on the tracker output: on Figure 7, we can clearly see that the children, originally depicted as 'unknown' for the tracker, since this was the class at the moment of their first appearance, are well-classified as no-mask throughout all the frames in which they are tracked.

This time, the depth estimation (once it has been smoothed) is quite precise as can be seen on the provided video or in from Figure 13 even if it is not perfect.

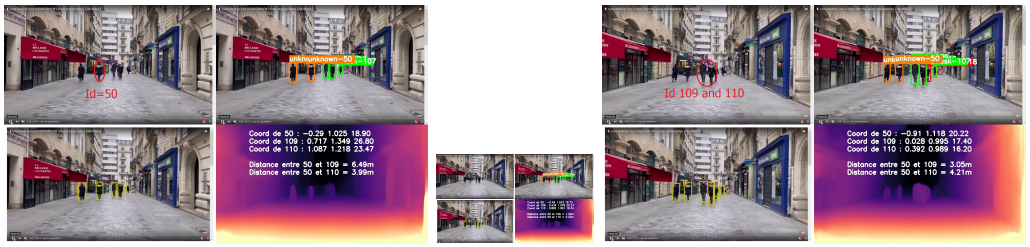


Figure 13: Frames 713, 753 and 783 showing person number 50 109 and 110

Still, if we had a few critics to make on our work, we should point the following :

- We can see on Figure 14 that there are small detection mistakes. This is the same problem than with President Sarzozy' tie. Again, a solution would be to add a a minimal number of frames the object has to appear in to be tracked.

Figure 14: Frame 9: Lamp detected as 'Unknown-3'

- The tracker is not very robust to occlusions. Indeed, in Figure 15, we can see a woman wearing a mask, hidden by somebody else, reappearing later: the tracker does not manage to follow her all along because she is occluded for too many frames. However, this example also demonstrates that the mask predictions are coherent, since when she reappears, a new tracker is instantiated and she is classified as 'unknown', which is very coherent since she is now back to the camera.

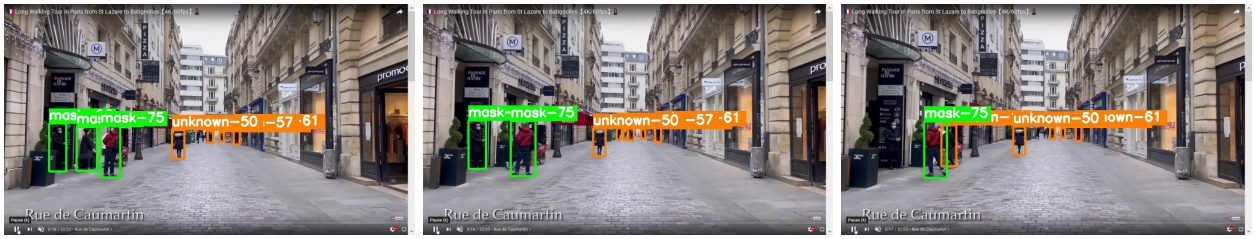


Figure 15: Frames 403, 417 and 427. From left to right : Woman in black detected as 'Mask', Woman in black occluded, same woman now detected as 'Unknown'

7 Conclusion

As a whole, our results show impressive results on these two videos, and we believe that by altering several hyper-parameters, the overall process is quite robust

Currently, our implementation does not allow for real-time detection because it is too slow: for debugging purposes, we save text files and images containing the information each each step and propagate these files through our implementation. As Input/Output are very time consuming, we believe that almost everything we have coded could be done in one pass... allowing for real-time use. However, if this was to be done, we could not back-propagate the correct class for an object through the previous frames like we do with our post-processing on the tracker. To remedy this, we could use a batch-tracker which would allow for almost real-time use, with only a couple of frames latency.

8 References

References

Alex Bewley, Zongyuan Ge, L. O. F. R. and Upcroft, B. [2017]. Simple online and realtime tracking.

Godard, C., Aodha, O. M. and Brostow, G. J. [2018]. Digging into self-supervised monocular depth estimation, *CoRR* **abs/1806.01260**.
URL: *http://arxiv.org/abs/1806.01260*

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. [2015]. You only look once: Unified, real-time object detection.