

TEC2017-88169-R MobiNetVideo (2018-2020-2021)

*Visual Analysis for Practical Deployment of Cooperative Mobile Camera
Networks*

D4

Deployment and application scenarios

Video Processing and Understanding Lab

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Supported by



AUTHORS LIST

José M. Martínez	josem.martinez@uam.es
Álvaro García-Martín	alvaro.garcia@uam.es
Marcos Escudero-Viñolo	Marcos.escudero@uam.es
Pablo Carballeira López	Pablocarballeiera@uam.es
Juan Carlos San Miguel	Juancarlos.sanmiguel@uam.es

HISTORY

Version	Date	Editor	Description
0.1	16/07/2021	José M. Martínez	First Draft version
0.2	12/06/2019	José M. Martínez	Contributions
0.3	24/07/2021	Álvaro García-Martín	Contributions
0.4	04/09/2021	Marcos Escudero-Viñolo	Contributions
0.5	11/09/2021	Pablo Carballeira López	Contributions
0.6	15/09/2021	Juan Carlos San Miguel	Contributions
0.7	26/09/2021	José M. Martínez	Final Working Draft – Editorial checking
1.0	28/09/2021		First version

CONTENTS:

1. INTRODUCTION	1
1.1. DOCUMENT STRUCTURE	1
2. APPLICATIONS	3
2.1. VEHICLE RE-IDENTIFICATION APPLICATION.....	3
2.1.1. <i>Introduction</i>	3
2.1.2. <i>Application</i>	3
2.2. APPLICATION FOR CONTINUAL LEARNING.....	6
2.2.1. <i>Introduction</i>	6
2.2.2. <i>Interface Description</i>	7
2.2.3. <i>Workflow</i>	7
2.2.4. <i>Advantages</i>	8
2.3. APPLICATION FOR THE AUTOMATIC REGISTRATION OF TRANSITED SPACES.....	8
2.3.1. <i>Introduction</i>	8
2.3.2. <i>Application</i>	9
2.4. OBJECT CLASSIFICATION AND DETECTION MOBILE APPLICATIONS BASED ON LIGHT CONVOLUTIONAL NEURAL NETWORKS	10
2.4.1. <i>Introduction</i>	10
2.4.2. <i>Development</i>	11
2.4.3. <i>Evaluation</i>	13
2.5. APPLICATION FOR DETECTION-AWARE TRACKING OF MULTIPLE OBJECTS.....	14
2.5.1. <i>Introduction</i>	14
2.5.2. <i>Framework</i>	15
2.5.3. <i>Included detectors and tracker models</i>	16
2.5.4. <i>Correlation metrics for evaluation metrics</i>	17
2.5.5. <i>Detectors with different performance</i>	17
2.5.6. <i>Evaluation</i>	18
3. REFERENCES	21

1. Introduction

This deliverable recapitulates the different applications developed within WP4 during the 9 months extension of the project (January-September 2021).

1.1. Document structure

This document contains the following chapters:

- Chapter 1: Introduction to this document
- Chapter 2: Description of the applications developed within WP4

2. Applications

2.1. Vehicle Re-Identification Application

2.1.1. Introduction

Vehicle Re-identification (Vehicle ReID) is a computer vision task whose relevance has been increasing during the last years due to the growing emergence of smart cities and Intelligent Transport Systems (ITS) that make use of this technology. For instance, it allows obtaining knowledge about the traffic flow, which makes it possible to adapt the traffic lights smartly or provide useful information to the autonomous driving field.

The main objective of Vehicle ReID is to identify a particular vehicle (query) recorded by a camera among a set of gallery images that have been recorded by different cameras. In particular, we have participated in the AI City 2021 Challenge (www.aicitychallenge.org), which asks the participants to provide a ranked list of 100 vehicles sorted according to their similarity to the query. Thus, ideally, the first images in the list should have the same identity as the query image.

However, the ReID task is composed of different challenges. Firstly, the different vehicle samples suffer from small inter-class variability due to similar orientation, colour or model, among other characteristics. In fact, similar backgrounds and orientations often generate a severe bias that reduces the distance between different vehicles. Moreover, at the same time, it exists a large variability between frames of the same vehicle owing to different illumination conditions, resolution, points of view... etc.

Due to these possible drawbacks, it is necessary to analyse the performance of the algorithms taking into account these challenges. For this reason, it is necessary to observe the visual results at the same time as the numerical outputs, which allows being aware of when the algorithm is making mistakes due to these difficulties. For example, it allows checking when the algorithm is identifying two vehicles erroneously because they have similar orientations. In addition, it makes it possible to analyze when the different proposals are solving these types of issues. Therefore, it is considered that the development of a demo that helps us to evaluate the algorithm in this way is very useful.

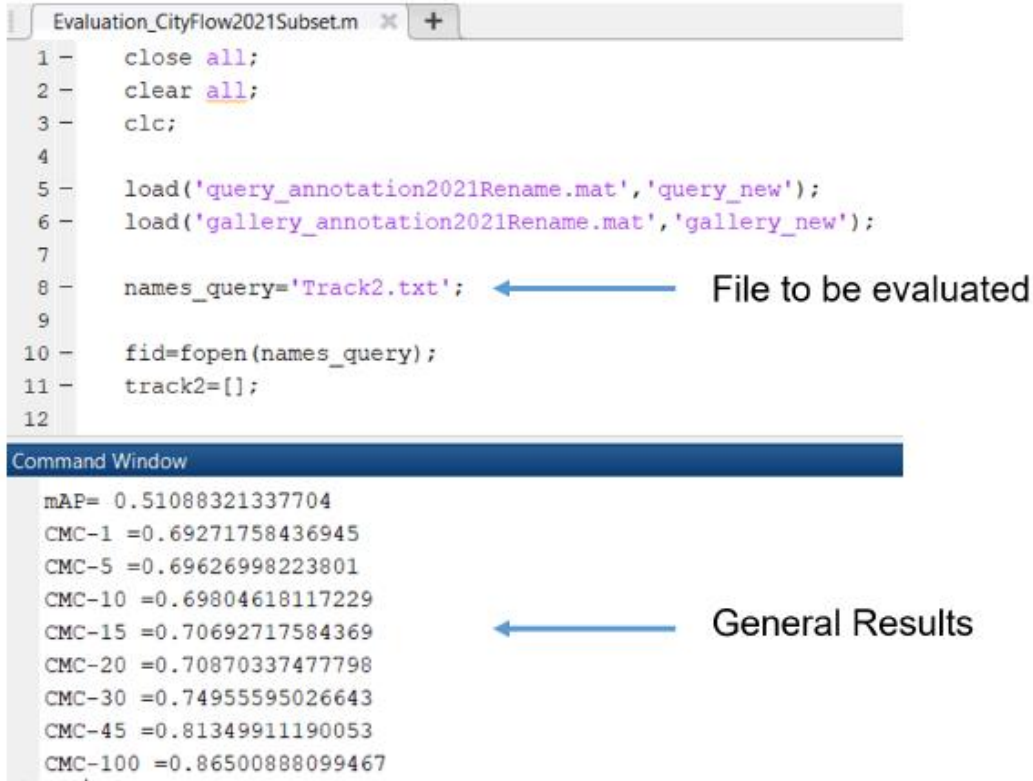
2.1.2. Application

The demo is divided into two main steps. Firstly, it is necessary to obtain the numerical results by using an evaluation code in Matlab. Then, a user interface developed in Python makes it possible to visualize the results visually and numerically.

As mentioned before, the first step to being able to make use of the proposed demo is to obtain the numerical results using Matlab. Figure 1 illustrates the appearance of the evaluation script in which we only have to indicate the file with the algorithm output we want to evaluate. Thus, by running this script, we obtain the general results that include the mean Average Precision (mAP) and the CMC-k with $k = [1; 5; 10; 15; 20; 30; 45;$

100]. Moreover, this script generates the necessary files to run the user interface. These files are:

- Text file that includes the AP result for each query and the general results.
- Text file for each query with the correct indexes in the obtained top100 ranking.



```

Evaluation_CityFlow2021Subset.m x +
1 - close all;
2 - clear all;
3 - clc;
4
5 - load('query_annotation2021Rename.mat','query_new');
6 - load('gallery_annotation2021Rename.mat','gallery_new');
7
8 - names_query='Track2.txt'; ← File to be evaluated
9
10 - fid=fopen(names_query);
11 - track2=[];
12
Command Window
mAP= 0.51088321337704
CMC-1 =0.69271758436945
CMC-5 =0.69626998223801
CMC-10 =0.69804618117229
CMC-15 =0.70692717584369
CMC-20 =0.70870337477798
CMC-30 =0.74955595026643
CMC-45 =0.81349911190053
CMC-100 =0.86500888099467
    
```

Figure 1. Matlab evaluation script: It is only necessary to indicate the file that we want to evaluate to obtain the mAP and CMC.

Once we have run the Matlab script, it is possible to use the user interface to be able to navigate through the visual results while viewing the numerical results. Firstly, it is necessary to run the command: Python visualize2.py in the terminal to start the interface. Thus, the window shown in Figure 2 appears on the screen. Here we must indicate in "Txt Dir" the directory where the results are located and press the "load" button to access the main window of the interface.

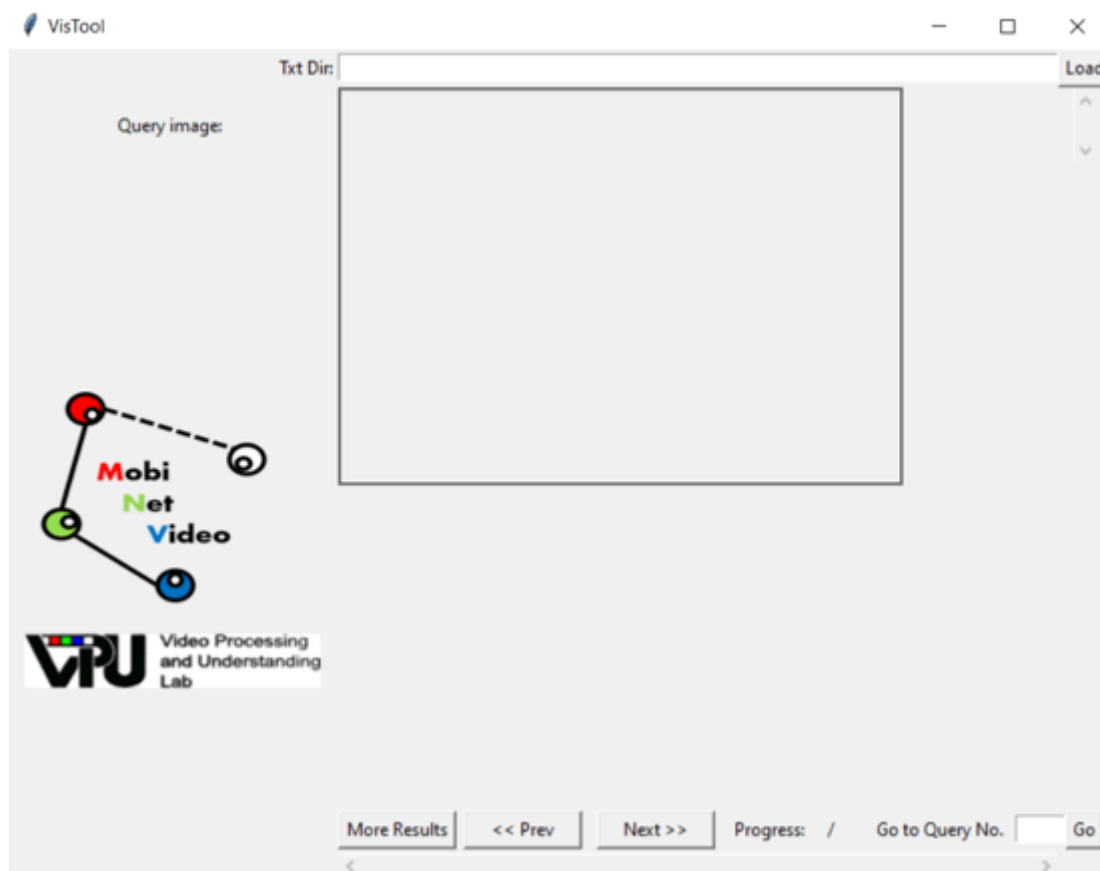


Figure 2. Python User Interface: Initial window. It is necessary to introduce the path where the results are located and press "load" to access the main window.

Once we pass the initial window, we arrive at the main window (Figure 3). It can be divided into different blocks. Firstly, on the left we find the query image while the rest of the images are the top100 ranking. Then, in the ranking, the correct images are indicated with a green bounding box while the incorrect ones are indicated with red. In addition, all the candidates have indicated their position in the ranking. Secondly, the top left corner shows the AP (Average Precision) for each query and the mAP, which is the average of the different AP and, therefore, it does not change. It allows us to always know the general result (mAP) while at the same time it allows knowing the particular result of each query and, therefore, to be able to evaluate them independently.

Furthermore, it includes a control panel to navigate through the queries. It makes it possible to go to the next or the previous query or to go directly to a specific one. This panel also has a "More Results" button. This button displays additional results (Figure 4) for deeper analysis of the algorithm performance. In particular, it shows different values of the CMC.

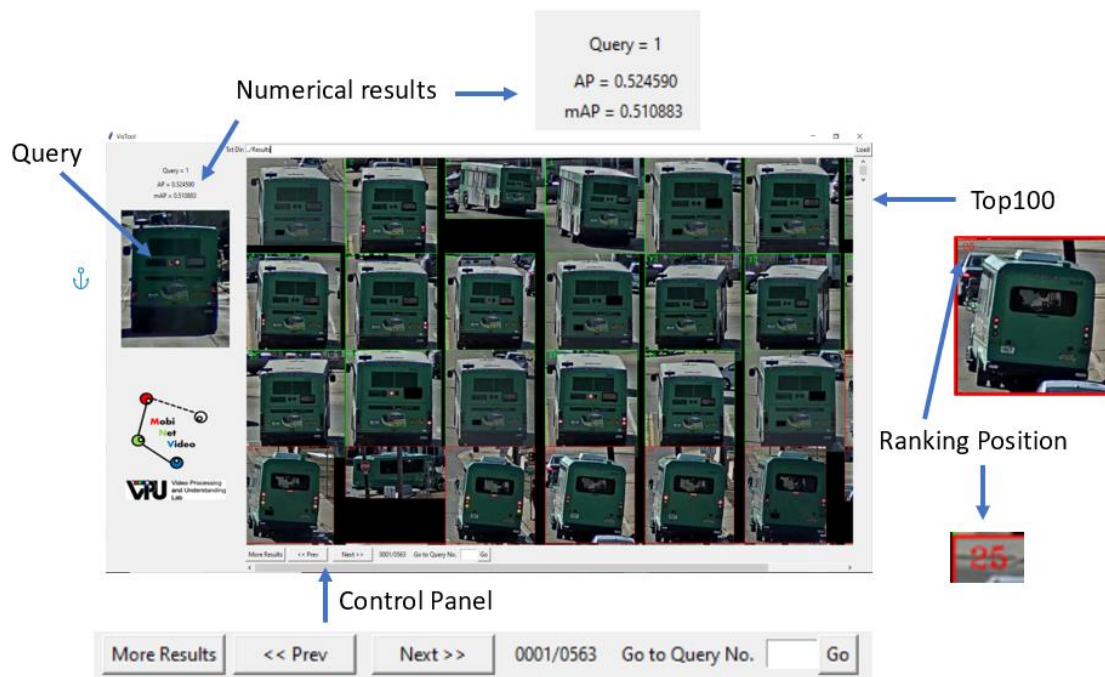


Figure 3. Python User Interface: Main window. It contains the query, the top100 ranking and the numerical results. Additionally, it includes a control panel that allows an easy navigation through the queries.

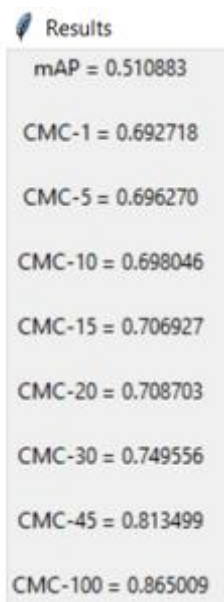


Figure 4. Python User Interface: Additional results.

2.2. Application for Continual Learning

2.2.1. Introduction

The application is contextualized in a Continual Learning workspace environment. The purpose of the interface is to allow the user to select different data and weights every time the application is run. This way, the user can train and test the model and see the results without using the console.

2.2.2. Interface Description

The interface is divided into four parts (Figure 5):

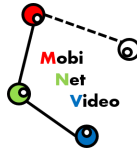
- **Data:** the “Data” area allows the user to select the data from a specific city to train the model. There is also the possibility to select various cities in order to combine the data. This makes it possible to use rehearsal techniques. To select a city, the user must click in the white square on the left of the desired city.
- **Weights:** this area allows the user to select specific initial weights to train the model. To do this, the user must click the “Select Weights” button and a File Explorer opens. Only “.pth” files can be selected. It is mandatory that the weights are compatible with the architecture.
- **Results:** this area is only shown after a training process has finished. Since the validation process is performed in every city, the user is allowed to select any preferred city to check the accuracy and the recall.
- **Train/Quit:** the “Train” button calls to the backend scripts which move the data and run the training process. The application requires selecting data and weights before running the training, otherwise the training will not be launched. The “Quit” button finishes the application.



Figure 5. Python User Interface and Results.

2.2.3. Workflow

1. The first step is selecting the desired data. Every time a city is selected, it is added or removed to/from a list that determines the subset that will be used. This



- is the subset that is moved (and combined if required) to a different directory when the “Train” button is clicked.
2. The second step is to select the weights. As explained, the user must know the location of the weights and they must fit the architecture. These weights may have also been obtained from a previous training.
 3. The third step is pressing the “Train” button. This automatically launches the entire backend process:
 - a. Data movement: the selected data is moved/combined into another directory, and a JSON file with the annotations is created.
 - b. Training: the model, initialized with the selected weights, is trained. The hyperparameters of the training process can be adjusted in a configuration file. The weights refreshed after every epoch are also stored in another folder.
 - c. Validation: the data from all the cities is processed with the weights obtained from the last iteration. As a result, the user gets a file (one for each city) with the metrics accuracy and recall.
 4. The metrics are displayed in the interface, highlighted in green, so the user does not need to search the file. The user can choose any city and its results will be displayed.
 5. The user can press the “Quit” button to switch off the application or restart the cycle at point 1.

2.2.4. Advantages

To sum up, the advantages of using the interface are:

- The user does not have to know how to use the console (although some messages are displayed on it).
- The user can select and combine the data from the available cities.
- The user can select the desired weights for model initialization.
- The user can automatically launch the training process pressing a button.
- The user can see the results without searching for the metrics files.

2.3. Application for the automatic registration of transited spaces.

2.3.1. Introduction

In recent years having a register of the places that a person has visited is a potentially very powerful tool for several tasks. For instance, it can even be used for tracing COVID-19 contacts, a ubiquitous topic in these times, or to model routines of a person to teach a machine. In this application, to better sense the areas attended by a person, ego-centric point of view (or lifelogging) scenarios are the preferred choice. Previous work on this is scarce so there is not an accurate and solid method in this vein. Ego-topo

was the first method reaching promising performance. The method is based on online constructing a graph with each node associated to an area (e.g., sink) in the place (e.g., kitchen) and then identifying the area at which each frame is captured by measuring similarities through a CNN between the stored areas representations and a video slot around the frame.

We have extended their approach, improving their results and incorporating new functionalities to their method before deploying it to a real application. In the first place, a study has been made of the systems that have led to the current Ego-topo. Next, the theoretical and algorithmic details of the Ego-topo system were delved into, with the aim of understanding its strengths and weaknesses. Later, several points of the system were modified in order to achieve added functionality, including enabling the use of external videos and the construction of combined graphs between different users. These combined graphs allowed the development of an application to detect contacts between two users in a domestic environment, with the aim of infectious diseases tracing. Before the application development, some weaknesses of the base system that harm its results were alleviated. The changes were evaluated first subjectively and then objectively with the creation of semi-automatic region annotations. An example of the created graph for a kitchen scene is included in Figure 6.



Figure 6. Graph description of the place kitchen in areas, in the example the sink area is detected as the place the egocentric camera of the person is capturing.

2.3.2. Application

The final purpose of the application is to accumulate statistics on whether two users have been in the same areas. If they have shared areas, we could denominate this as a direct or indirect contact, depending if the region is shared simultaneously or in different moments.

The combined graphs functionality allows to know if two users have been in the same region. A visit added to a previous node means that both users have passed by there. If we indicate the exact time at which each video begins, we can also determine if they have matched the regions simultaneously or not.

The process can be sketched as follows:

- We get an initial graph with the regions presented in the place to be analyzed.
- With this initial graph we got a combined graph for each input video (coming from different users).
- We are going to assume that two users enter the place at the same time, and we will measure if they have been in the same areas, and if they have been in them at the same time, showing in that case, how long they have remained together.

depicts an example of the graphical user interface.

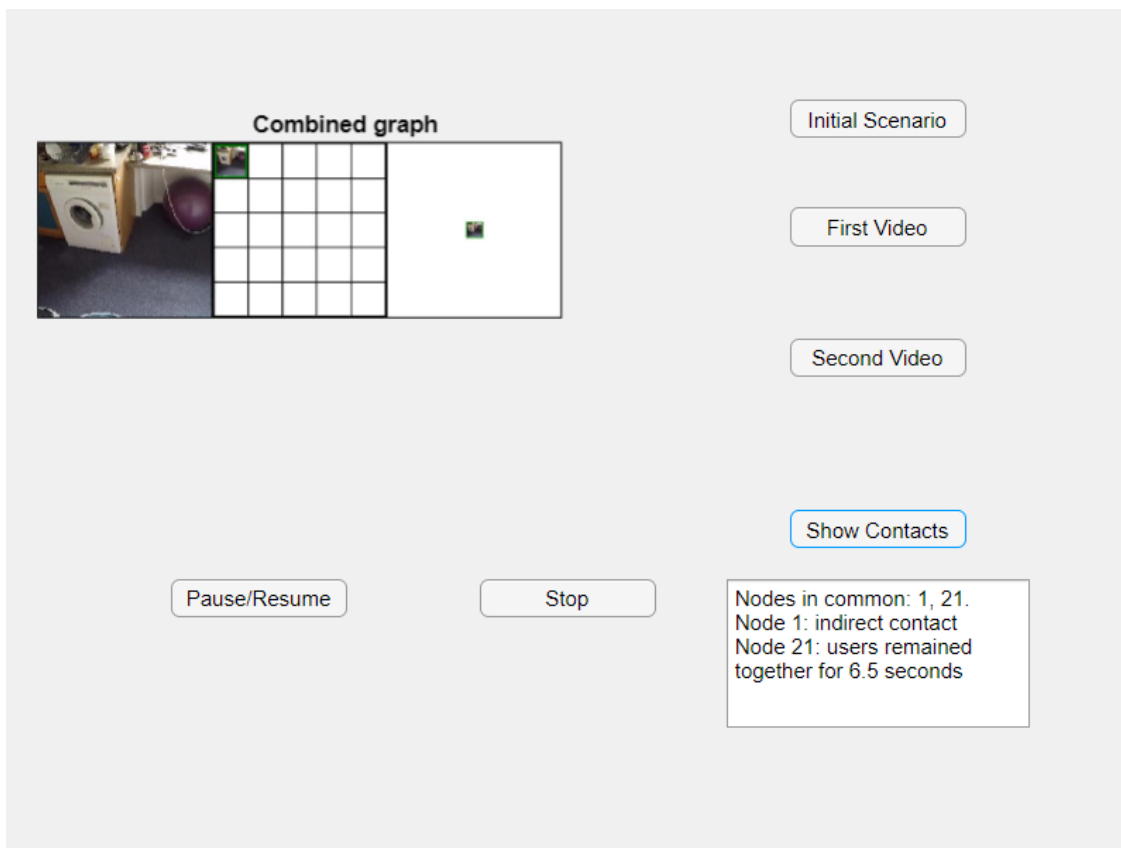


Figure 7. User Interface of the contact tracing application.

2.4. Object classification and detection mobile applications based on light Convolutional Neural Networks

2.4.1. Introduction

In recent years, computer vision has evolved rapidly due to the development of convolutional neural networks (CNNs), which have enabled an impressive boost in the performance of several applications that use visual information, such as: image

classification, object detection, or image segmentation. However, integrating deep-learning-based computer vision models in mobile devices is a complex problem due to the computational requirements of CNNs. Lightweight CNNs target this problem by focusing on the development of memory-efficient and low computational complexity. This work, which is further developed in the corresponding Master Thesis [1], is oriented to the development of mobile applications for the tasks of image classification and object detection using lightweight convolutional networks and quantized models, reducing their size and computational complexity.

2.4.2. Development

The core deep learning programming framework that has been used for the integration of the CNN models in the mobile applications is TensorFlow Lite. TensorFlow Lite is currently in constant development and plays a fundamental role in the integration of deep learning models in lightweight devices. The development of the mobile applications has been oriented to enable the use of different pre-trained CNN models that need to be previously converted into the TensorFlow Lite format. Thus, in addition to the design and implementation of the mobile apps, the work has been oriented to compile a framework to convert a CNN model into TensorFlow Lite format and later allow the insertion of the models into mobile devices with Android operating system using developer tools.

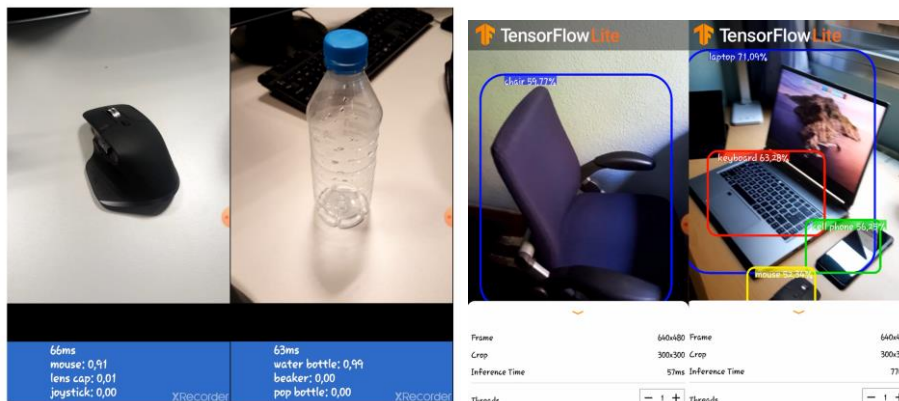


Figure 8.Examples of the developed image classification and object detection applications

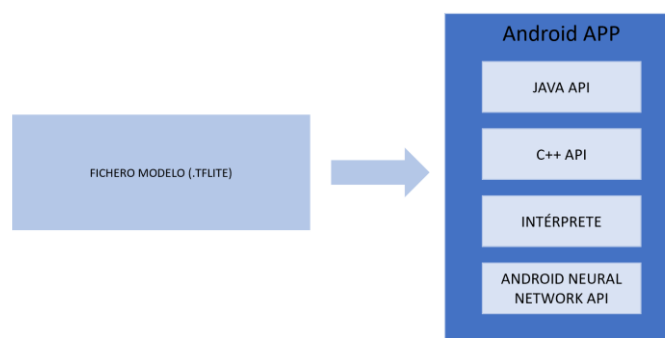


Figure 9. Overview of the application architecture (right), in which a pre-trained model in the TensorFlow Lite format can be integrated (left)

Figure 8 depicts an example of the implemented classification and object detection applications in action. **Figure 9** shows an overview of the software architecture used to implement the applications. The Android Apps are implemented using a Java API wrapper class that handles the control of the camera, image capture, etc. and wraps a C++ API, which is responsible of loading the Tensorflow Lite kernel and the pre-trained model which is used to perform inference.

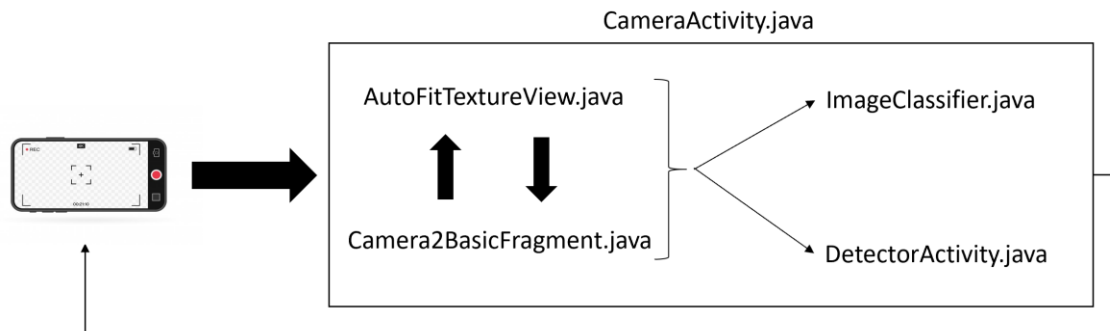


Figure 10. Diagram of the interaction of the Java scripts that are used to implement the mobile applications.

Figure 10 shows the diagram of the functions and interaction of the Java scripts that are used to implement the mobile applications. The main functions of the scripts is overviewed here:

- CameraActivity.java, is in charge of creating the graphical interface of the app, and integrates the functions of the rest of the scripts.
- Camera2BasicFragment.java, contains the class that manages the mobile camera to capture images that are later processed by the classification or detection scripts (ImageClassifier.java or DetectorActivity.java)
- AutoFitTextureView.java, continuously executes the re-sizing function on the images that are captured, and shows them in the graphical interface of the app.
- ImageClassifier.java, integrates with the TensorFlow platform to execute the TFLite classification models on captured images. It loads the TFLite interpreter and shows the 3 top-score classes and inference time in the graphic interface. The application can be configured to used different TFLite models by changing the path pf the model file and an assets folder that contains the list of class labels used to train the model.
- DetectorActivity.java, operates similarly to ImageClassifier.java integrating the TensorFlow platform and executing object detection classification models on captured images. The script displays the bounding boxes of detected objects with score over 0.5. Similarly to ImageClassifier.java, the application can be configured to use different TFLite models by changing the path pf the model file and an assets folder that contains the list of class labels used to train the model.

2.4.3. Evaluation

In addition to the development of the apps, the work has been focused in a comparative evaluation of different image-classification/object-detection in terms of both classification/detection performance, and computational complexity. Moreover, the influence of different levels of quantization in the parameters of the models has been evaluated. Some representative results in the performance and computational complexity of both applications are shown in the following Figures.

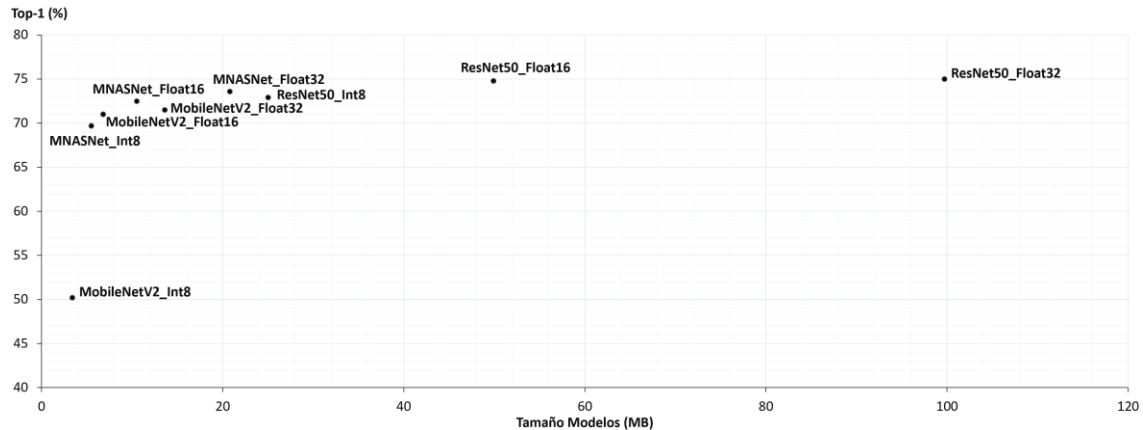


Figure 11 Top-1 classification accuracy for different classification models and levels of parameter quantization. X-axis represents the size of the model (in MBs) and the Y-axis the top-1 performance in a subset of the ImageNet validation set.

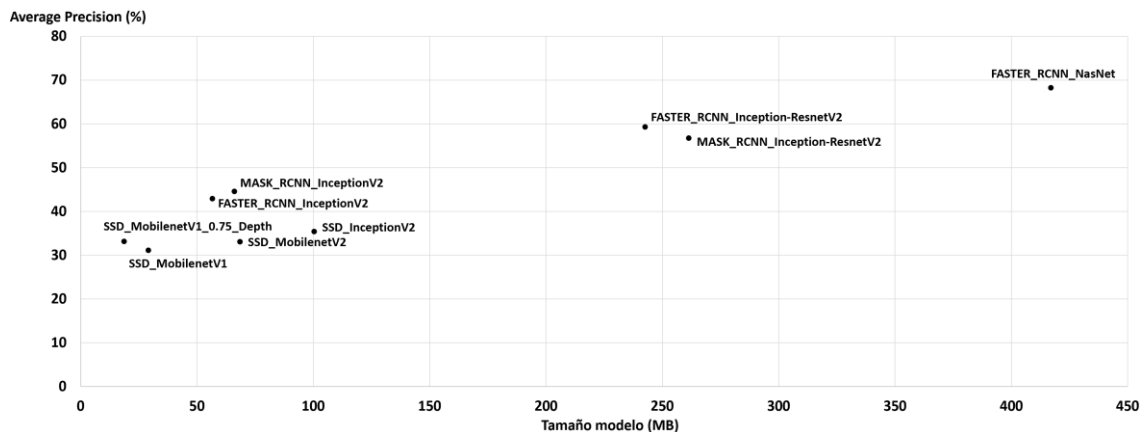


Figure 12 Average Precision different object detection models and levels of parameter quantization. X-axis represents the size of the model (in MBs) and the Y-axis the Average Precision in the COCO dataset.

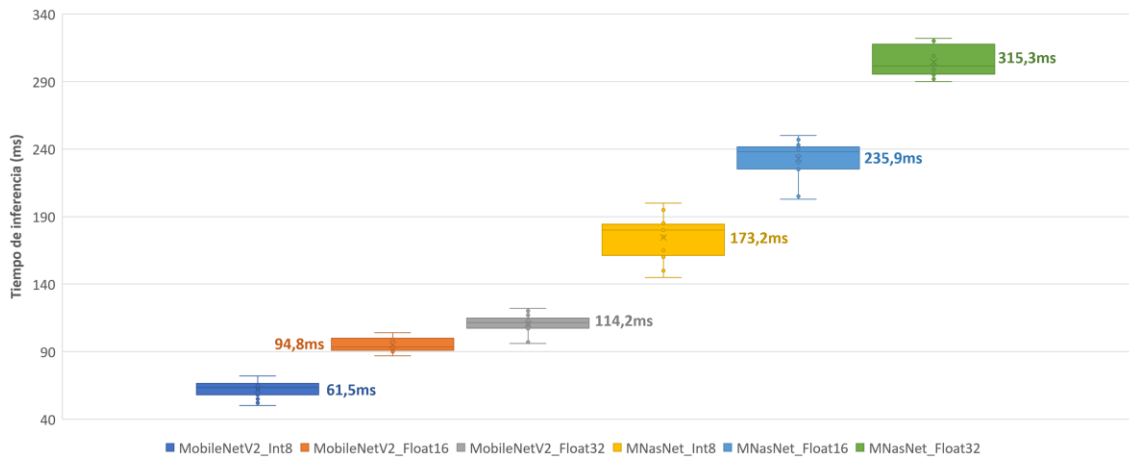


Figure 13 Evaluation of the Computational Complexity of different image classification models and parameter quantization levels. The computational complexity is measured by average classification inference times in a Samsung Galaxy S6 Edge+.

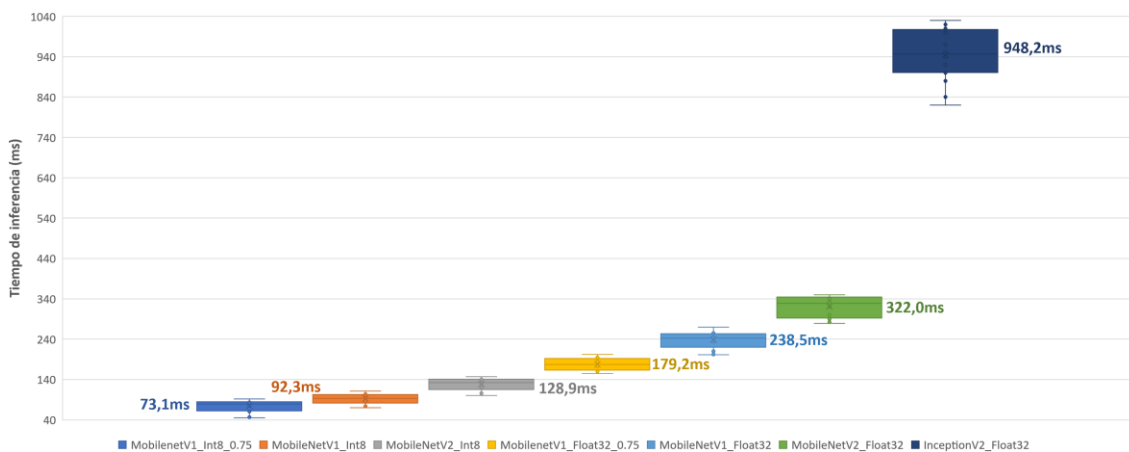


Figure 14 Evaluation of the Computational Complexity of different object detection models and parameter quantization levels. The computational complexity is measured by average classification inference times in a Samsung Galaxy S6 Edge+.

2.5. Application for detection-aware tracking of multiple objects

2.5.1. Introduction

Multi-Object Tracking (MOT) is a hot topic in the computer vision field. It is a complex task that requires a detector, to identify objects, and a tracker, to follow them. It is useful for self-driving, surveillance and robot vision, between others, where research teams and companies are trying to improve their models. In order to determine which model performs better, they are scored using tracking metrics.

Here we explore some of the design decisions to create the work environment, as well as list the selected the models and metrics to use during the experiments. Experimenting with metrics requires real data. To simplify the obtaining of results we have created an environment in which we can combine multiple detectors, tracker and metrics. As discussed later in this section, the framework facilitates the implementation of different models, looking for an optimal way to execute and join them.

This environment has been written in Python. The code can be found at the following link: <https://github.com/JorgeMunnozAguado/MOT-experimental-framework>

2.5.2. Framework

The proposed framework allows trackers to use bounding boxes from available detectors and evaluate outputs from any model. To accomplish this objective, we design the framework as show in the following figure. Here we explain some design details.

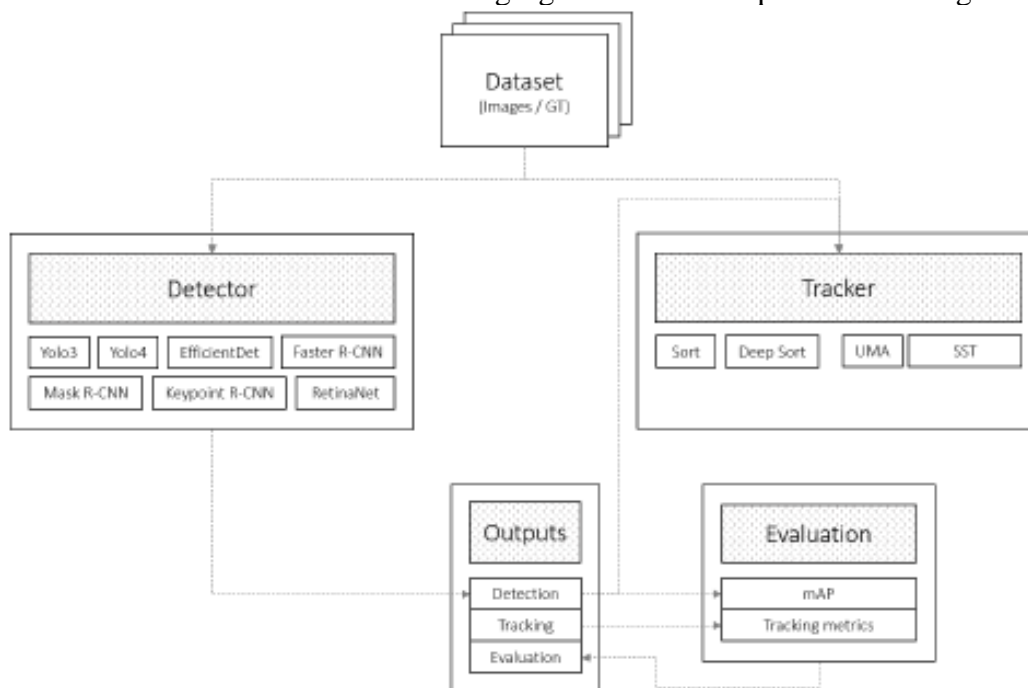


Figure 15 Design architecture of the experimental framework for MOT tracking.

The system is composed of these main modules: dataset, detectors, trackers, evaluation, output and auxiliary programs that help with debugging and testing. Each module is explained below:

- Dataset module controls the ow of the data for testing and training purposes. All sets are stored in a folder called dataset/ and must follow the same structure. Each set must contain the images (in a folder called img1/) and the ground truth (in the folder called gt/gt.txt).
- To simplify the integration of detectors and trackers we design two encapsulating classes for detection and tracking. Both classes contains the basic functionality, helping the integration of the models in an efficient way. Also this design decision helps to run the models with different configurations, weights and hyper-parameters.

- Outputs of detection and tracking models must be stored in a folder. We use a hierarchy in such a way that we distinguish the results executed by each detector and tracker.
- Evaluation requires to access the outputs. As shown in the figure, evaluation have multiple sub-modules. Detection and tracking tests use different metrics, each sub-module is specialized in evaluating detection or tracking.

In the framework we also include some useful tools to facilitate the obtaining and analysis of the outputs. A visualization tool was added to debug the models and generate the images. It could be used to generate videos or figures for multiple purposes. To make it easy to install and run the framework we added a setup feature. Running the setup script will download the datasets, create all environments, and download the models, if needed.

2.5.3. Included detectors and tracker models

The environment, as already mentioned, is mainly made of detection and tracking models. The framework allows to easily combine both. As already explained, model integration consists on creating a subclass for a model, from an abstract class. Thus, all models can be executed from a central main for detection and another for tracking. This simplifies the methods for testing. In this subsection, models included in the environment are listed. For ease use, we included pre-trained models as default. There is also the possibility to train or fine-tune models.

The detection and tracking models included are those found in the following Figures:

Detectors		
Name	Brief description	Year
Faster-RCNN	Two-step model with a Region Proposal Network propose possible objects and a classifier decide if they are or not.	2017
Yolo V3	One-step model. Propose regions with anchors.	2018
Yolo V4	Improvement over Yolo V3	2020
RetinaNet	Uses a feature pyramid propose regions where objects can exist.	2017
Mask R-CNN	Similar to Faster-RCNN, but also segments the image.	2017
Keypoint R-CNN	Combine local keypoints with Faster R-CNN technology.	2020
EfficientDet	Uses a special architecture called BiFPN.	2020

Figure 16 List of analysed detection algorithms.

Trackers		
Name	Brief description	Year
Sort	Uses the position of objects to predict and track them.	2016
Deep Sort	Like Sort, it makes use of the trajectory and includes a new method based on visual characteristics to differentiate objects.	2017
UMA	Uses an architecture based on siamese network to distinguish objects.	2020
DAN	Extracts descriptors from a multi-spacial backbone pyramid to track objects.	2018

Figure 17 List of analysed tracking algorithms.

2.5.4. Correlation metrics for evaluation metrics

The correlation matrices are created comparing metrics by combining scores from sets and models included for the specific test [2]. Process starts by selecting all scores from two metrics. These two lists of scores are compared by the Pearson product-moment correlation coefficient. This product returns a value in the range -1 to 1, indicating how correlated are they. Negative indicates that it is inversely correlated. Each of the boxes from the matrix was built by this way. The diagonal has the value 1 since we are comparing the same metric. Half of the matrix is a mirror of the other half, because of the way it has been constructed.

2.5.5. Detectors with different performance

To run the tests, we decided to use detectors with different performances. We seek to observe the work of the tracker when the quality of the detections varies. Having detectors with different performances allows us to check how the tracking metrics behave in this situation. And check the correlation of the tracking metrics with the detection ones.

In order to achieve the most reliable results, we used for experiments different versions of Faster R-CNN model. This allows us to have a homogeneous variety of detectors in terms of performance. Thus, having detectors with different accuracy allows more stable results to be achieved. Modifications to the models have been achieved by varying their hyperparameters. In the following Table you can find the modified values in addition to the performance in mAP, to check performance gap between models.

Detectors			
Name	box_score_thresh	box_nms_thresh	mAP
Faster-RCNN (<i>default</i>)	0.05	0.5	55.31
Faster-RCNN mod-1	0	0	36.87
Faster-RCNN mod-2	0.5	0.5	49.32
Faster-RCNN mod-3	0.9	0.9	31.34
Faster-RCNN mod-4	0.99	0.5	18.15
Faster-RCNN fine tune public	0.05	0.5	72.47
gt	-	-	49.93
			100

Figure 18 List of detectors used for experiments.

We also include ground truth and public detections to feed the 100% performance gap. Ground truth detections must have a 100% of accuracy, as the bounding boxes were extracted from the ground truth files. MOT Challenge includes in their datasets public detections. These are detections that everyone has. They are used to compare the efficiency of the trackers using the same detections. Where *box score thresh* is the score allowed for a detection to be an object or not. As less threshold more detection will be. On the other hand, *box nms thresh* relate with non-maximum suppression. This technique allows the detector to remove duplicate bounding boxes over the same object. A high value in this variable means that there will be more bounding boxes.

2.5.6. Evaluation

The correlation matrices are generated with the data obtained from 11 sequences, from two datasets (MOT17 and MOT20 [2]), 8 detectors and 4 trackers, with a total of 352 possible combinations. In this section we experiment with 6 detection metrics and 6 tracking metrics, making a total of 12 metrics.

At start point, we show the correlation matrix including all the trackers, detectors and datasets in the following. The correlation matrix includes detection and tracking metrics to compare. Let's explore detection metrics.

As previously commented, Average Precision or AP is the most used detection metric. The metric is calculated as the area under the recall / precision curve, check Equation 2.1. mAP is the mean of AP for a set of sequences. As we can notice, mAP and recall are highly correlated. Despite, precision is hardly related to mAP and negatively related to recall. This makes sense since precision and recall are different things.

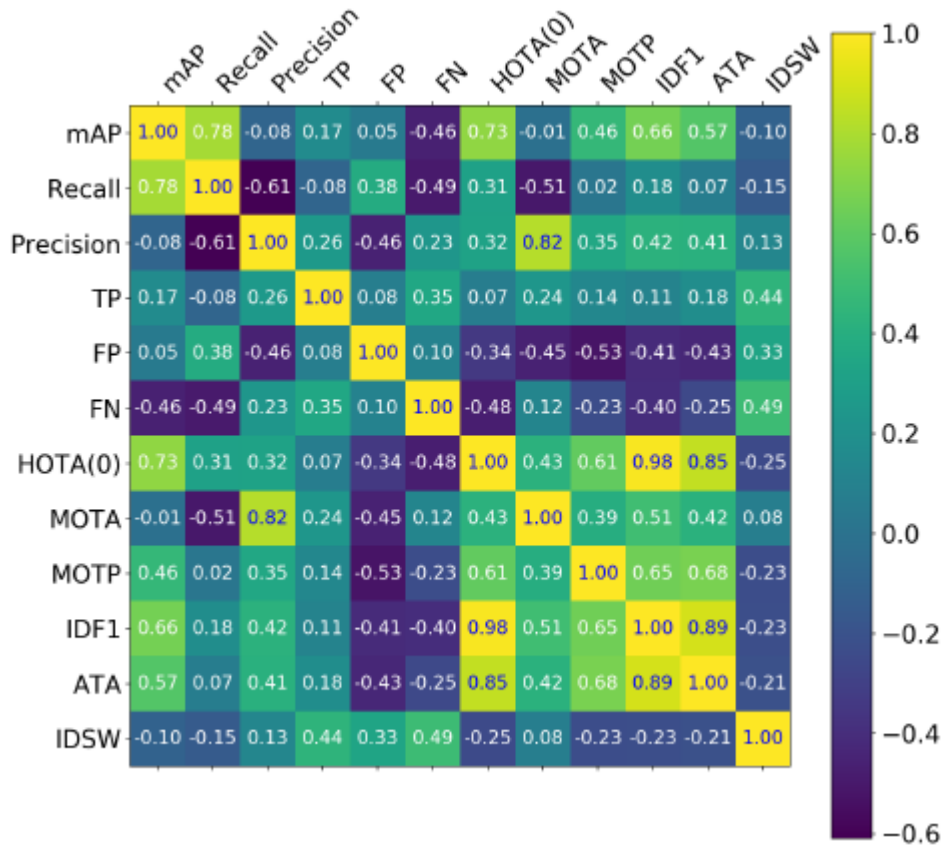
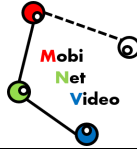


Figure 19 Correlation matrix for detection and tracking metrics using selected detectors and trackers over MOT17 and MOT20 datasets.

Other interesting features we can notice from Figure 5.1 is the relation of the detection metrics with TP, FP and FN variables. In the first place, mAP and recall are negative related to FN. Similar with precision and FP. Checking the formulas we end with an answer. Both FN and FP are in the denominator of recall and precision respectively.

Once the correlation between detection metrics has been studied, we proceed to analyze the tracking metrics. There are multiple metrics used in this context. First noticeable feature is that HOTA, IDF1 and ATA are high correlated between them. By analyzing the equations of the metrics, we can notice that are trace-based metrics. Our suspicions were confirmed in previous publications [2], where they commented that IDF1 and ATA are very similar with the difference that ATA measures the fraction of correct tracks. On the other hand, HOTA uses a double formula to evaluate detection and tracking separately.

Between tracking metrics we can observe other relevant characteristics. MOTP seems to be more correlated with IDF1 and ATA than with MOTA, as MOTP measures the average localization accuracy over the TP set. Identity switches (IDSW) measures the number of switches between objects identifiers (IDs) between two frames. Included as component of MOTA, shows in the results that it is inversely correlated with the rest of



the tracking metrics, even if it has not been calculated directly. Means that IDSW is an important feature to measure tracking performance.

Between detection and tracking metrics we can be found other relevant characteristics. False Positive (FP) variable is negative correlated with all tracking metrics, but IDSW. False Negatives (FN) are also negative related with HOTA, IDF1, MOTP and ATA. In view of the results MOTA barely has relation with FN, unlike IDSW with a high positive correlation with this variable.

MOTA metric also has other relevant features to consider. In the Figure we can notice how this MOT metric is highly positively correlated with precision and negatively related to recall, but not with mAP. This can be problematic as improving the precision of the detector would notably increase the tracking score. The issue of MOTA with precision was previously discussed in HOTA paper [3]. Other metrics as MOTP, ATA and IDF1 seems to have the same problem with the relation to precision. Despite this, in this topic HOTA is performing as expected, but it is high correlated with mAP. This is a problem, since the metric being related to detection can greatly vary the tracker score depending on the performance of the detector. Thus, not only the performance of the tracker would be evaluated.

3. References

- [1] Desarrollo de aplicaciones móviles de clasificación y detección de objetos a partir de redes convolucionales ligeras (Development of mobile application for object classification and detection based on light convolutional networks), Paulo C. Casa Robles (advisor: Pablo Carballeira López), Trabajo Fin de Máster (Master Thesis), Master en Ingeniería de Telecomunicación, Univ. Autónoma de Madrid, Jun. 2020.
- [2] L. Cehovin, A. Leonardis, and M. Kristan, "Visual object tracking performance measures revisited," *IEEE Transactions on Image Processing*, vol. 25, pp. 1261-1274, 3 2016.
- [3] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taix e, and B. Leibe, "Hota: A higher order metric for evaluating multi-object tracking," *International Journal of Computer Vision*, vol. 129, pp. 548{578, 2 2021.