COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

MASTER'S THESIS

INTERPERSONAL DISTANCE MONITORING USING DEEP LEARNING AND COMPUTER VISION

Author: Rodrigo López de Toledo Soler

Supervisors:

Jaime Boal Martín-Larrauri

Álvaro Jesús López López

Madrid, July 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título **Interpersonal Distance Monitoring Using Deep Learning and Computer Vision**

En la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico **2020/2021** es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Rodrigo López de Toledo [illegible] Fecha 09/ 07/ 2021

Autorizada la entrega del proyecto

Jaime Boal Martín-Larrauri

Alvaro Jesús López López

Fdo.: [illegible] Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H Fecha: 2021.07.09 20:33:39 +02'00' Fecha: 09/07/ 2021

Fdo.: [illegible] Fecha: 09/ 07/ 2021

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 12 de Julio de 2021

ACEPTA

Fdo. 53850940M RODRIGO LOPEZ DE TOLEDO

Digitally signed by 53850940M RODRIGO LOPEZ DE TOLEDO
Date: 2021.07.12 21:30:24 +02'00'

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

MASTER'S THESIS

INTERPERSONAL DISTANCE MONITORING USING DEEP LEARNING AND COMPUTER VISION

Author: Rodrigo López de Toledo Soler

Supervisors:

Jaime Boal Martín-Larrauri

Álvaro Jesús López López

Madrid, July 2021

Interpersonal Distance Monitoring Using Deep Learning and Computer Vision

Author: López de Toledo Soler, Rodrigo
Supervisor: Boal Martín-Larrauri, Jaime; López López, Álvaro Jesús
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

In these weird times in the last year and a half, keeping a safety distance between individuals to prevent Covid-19 from spreading has become one of the most effective measures to control de pandemic. The objective of this project is to determine if it is possible to monitor that distance using a deep learning approach that provides independence from camera position and if it can be done in real-time. After training different models for detection, tracking and distance estimation between individuals, results show that, with some improvement, this objective can be achieved within the limits of real-time video.
Keywords: Deep Learning, Detection, Tracking, Distance Estimation, Real-Time

1. Introduction

Since Artificial Intelligence and Machine Learning came to our lives, quite recently in fact, several fields of investigation and research branched out from it. As happened with this new "science" in other fields like, for example finance, the use of images and videos for further analysis and development was one of the main fields of interest, and therefore, computer vision was born.

Computer vision is an area of study that is identified as a subfield of artificial intelligence and works in parallel with Machine Learning. It makes use of specialized methods and general algorithms of Machine Learning. The goal of computer vision is the understanding of the content of the digital images and therefore, videos, to mimic the ability of the human vision. For us humans, understanding an image is very easy, it comes natural and effortless, but for a computer it is extremely complicated because it is very difficult to understand the changes in lighting, position, orientation and expression that we see so natural. In the early 1970s, computer vision was only an ambitious point in the agenda of giving robots a human intelligence. In the 1980s and 1990s, the focus shifted, researchers centered their efforts on more complex mathematical techniques to perform image analysis.

It is important to understand that image processing using common techniques such as image and contrast adjustment, histogram equalization and binarization is not computer vision. These techniques are very useful to apply to computer vision, but they are not computer vision because we cannot extract any analysis or knowledge from what is happening in the image.

Since the year 2010 (bear in mind that computer vision was born not so long ago), there have been several tasks in the field that have been extensively researched and that have achieved success. There were a lot of simple tasks that were widely researched and achieved great success. All these tasks, developed by making use of Machine Learning algorithms and image processing tools, were object detection and recognition, object classification and object segmentation. These simple tasks become tools for applications developed later and proven useful in the real-world. Some of them have been retail, for automated checkout, medical imaging, surveillance, biometrics and motion capture, being the last one the focus of this thesis.

2. Definition of the project

The project is primarily based on three different phases that, once completed, will work together and the ultimate goal of all three is to provide an accurate estimate of the distance between two individuals in a video. The objective is to make the system work in real time, which is the key to the correct operation of this project and the fulfillment of its objectives.

For the first part, detection, several models based on the algorithm par excellence in terms of real-time detection, which is YOLO [1], have been tested. Versions 3, 4 and 5 have been tested with different variations in both hyperparameters and model size. For these versions, each model has been trained in different datasets and its performance has been evaluated in terms of detections and processing speed, using FPS (Frames Per Second). The datasets in question have been COCO[1], which is the quintessential dataset for detection tasks, KITTI[2], which is a dataset designed for autonomous vehicles, whose objective is to see if it is capable of generalizing other tasks, and CrowdHuman[3], a dataset specifically prepared for people detection tasks since all their annotations are human. Regarding the decision of the models to use, after doing an analysis to analyze the differences between models based on RCNN [2] (Region proposal frameworks) and models based on regression, such as YOLO, it was decided to use models based on regression since, although the precision is somewhat lower, the speed when processing the images is much higher since they are based on a single pass of the algorithm for each image, instead of generating many possible regions and then comparing them. Due to that, the precision decreases, but the speed increases, and, taking into account that the greatest limitation is real-time, it is more beneficial to use a regression-based model. In addition, by putting a tracking algorithm behind it, it will serve as a safety net against possible detection errors or losses. As mentioned above, the chosen model was YOLO and its different versions.

The second stage of the project is the follow-up. As discussed above, this part is a safety net after detection. What is sought at this stage is that, in the event that the detection of a person is lost in a frame, the system is able to re-identify it if it reappears. This is very useful for when occlusions occur between people, for a task such as monitoring the distance between people it is key to be able to manage these occlusions and that the system can both predict the position of a detection and that it can re-identify it. To achieve this important characteristic, the most recognized algorithm in this field has been used, which is DeepSORT [3]. This algorithm is an evolution of the SORT algorithm, which is based on Kalman filters so that the system has memory and can save information from past detections. What this algorithm allows is that, once a detection has been made, its information can be saved and a few frames later recovered. For example, if a person passes behind a group of people, the algorithm saves the information of speed, appearance and size, among other characteristics and, once that person appears again after leaving the group, the algorithm is able to identify it as the same detection prior to

1 https://cocodataset.org/#home

2 http://www.cvlibs.net/datasets/kitti/

3 https://www.crowdhuman.org/

occlusion. What DeepSORT adds is a fundamental difference in appearance puller. This part is in charge of comparing the detections before and after the occlusion. SORT [4] does this by measuring distances with well-known algorithms (Hungarian, Cosine distance…). DeepSORT changes the focus of this part and what it does is create a convolutional network that generates an appearance vector based on the features extracted from the images. That appearance vector is what is used to compare before and after occlusion. In this way, there is a much more robust system based on deep learning that provides more precise monitoring.

Finally, we move on to the phase of distance estimation. For this type of task, several solutions have been developed that seek this objective. However, none of these solutions are based on deep learning. Most are based on establishing previous references and comparing the position of each detection based on that previous reference. The closest thing to deep learning is changing your perspective from a bird's eye view. In this way the camera perspective is zenith [5] and there is no problem with the depth of the image. This has a problem, and that is that you have to select before monitoring the detections on which you want to measure and monitor the distance. This prevents the distance between everyone in the scene from being monitored, and new people entering the scene cannot be monitored. In order to monitor all that, a deep learning-based approach is implemented. As there are no datasets with distance annotations, a synthetic dataset has been generated with a simulator where, iteratively, two people are generated on a stage, the distance between them is measured and the stage is extracted as an image, the depth information as another image, but in black and white, and the distance measurement. All of that is put together in a dataset and used as input to the model. Two models have been used in this phase to be able to compare performance and results. One of them is a model based on convolutional networks that consists of 4 convolutional layers with their corresponding dropout to avoid overlearning, with a neck consisting of a fully connected layer of 30 neurons followed by another of 15. Finally, the output is a only neuron with linear activation function whose output is the estimate of the distance between the people in the image. The other model is a model based on DNNs. To be able to use it, the information of all the detections of the previous dataset has been put in the form of CSV, including the coordinates and dimensions of the detections, the depth value of those detections and, as a target, the distance.

3. Description of the model

As outlined in the chart below, the project follows the three stages proposed with some intermediate steps. The chart below depicts the inference of the project proposed, but all the models involved (Detection, Tracking and Distance Estimation), and some more, have been previously trained on different datasets and I have chosen the best performing ones for each stage, as described in the next section.

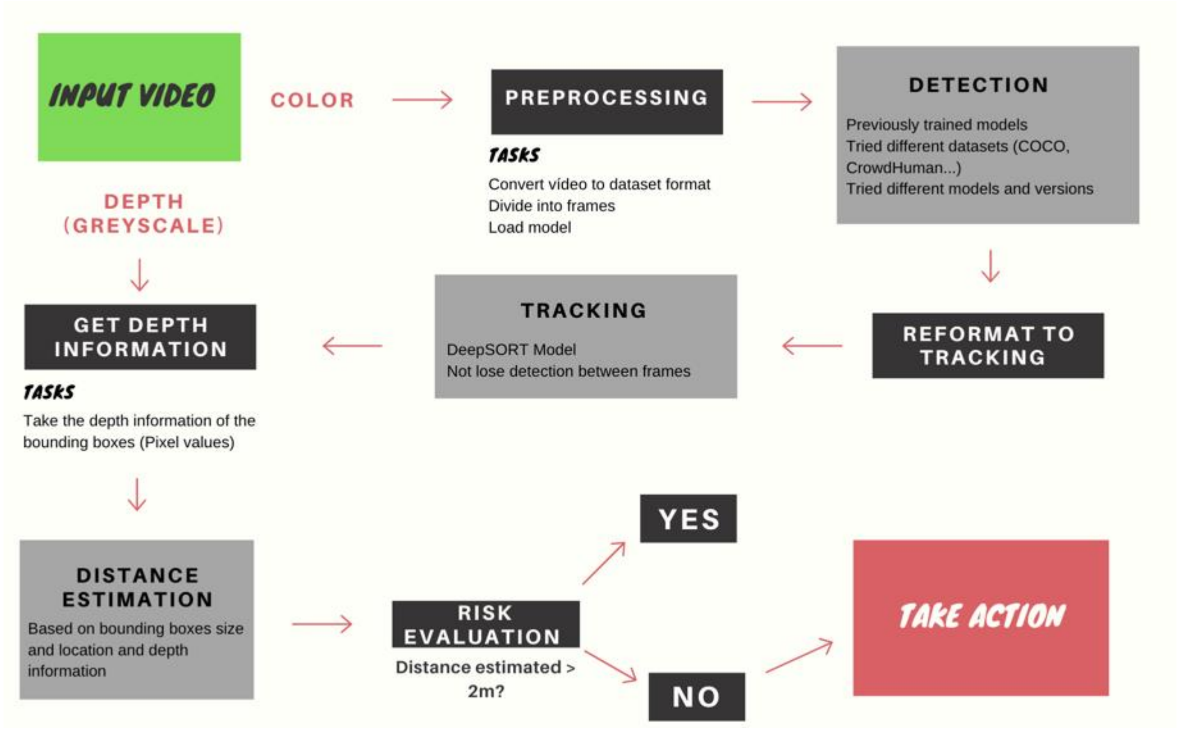

INPUT VIDEO
COLOR
PREPROCESSING
TASKS
Convert video to dataset format
Divide into frames
Load model
DETECTION
Previously trained models
Tried different datasets (COCO, CrowdHuman...)
Tried different models and versions
DEPTH (GREYSCALE)
REFORMAT TO TRACKING
TRACKING
DeepSORT Model
Not lose detection between frames
GET DEPTH INFORMATION
TASKS
Take the depth information of the bounding boxes (Pixel values)
DISTANCE ESTIMATION
Based on bounding boxes size and location and depth information
RISK EVALUATION
Distance estimated > 2m?
YES
NO
TAKE ACTION

Illustration 1 – Flowchart of the project

4. Results

In this section the results achieved by the models in each different stage will be presented and commented accordingly.

Detection:

- **Detection**: For this phase, the best performing model is YOLOv5 [6] trained on the CrowdHuman data set. The training results (of different size models) are shown below:

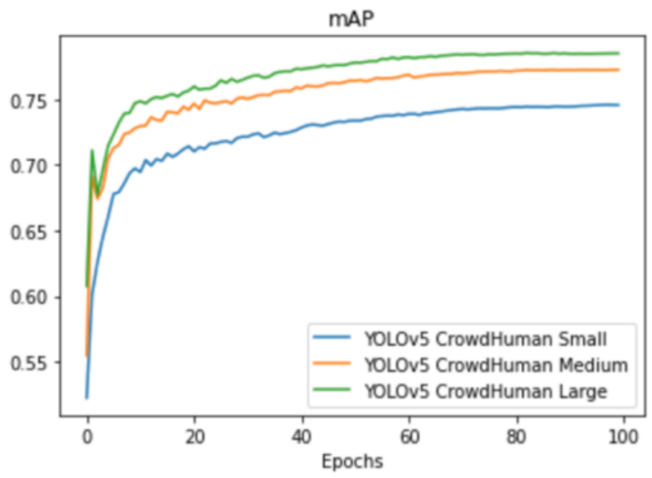


Illustration 2 – Training results on CrowdHuman and YOLOv5

For this phase, although the best results are with this model, more models have been tested that are worth commenting on. YOLOv3 [7] has been tested with the three datasets mentioned above, reaching a fairly good mAP in both KITTI and CrowdHuman.

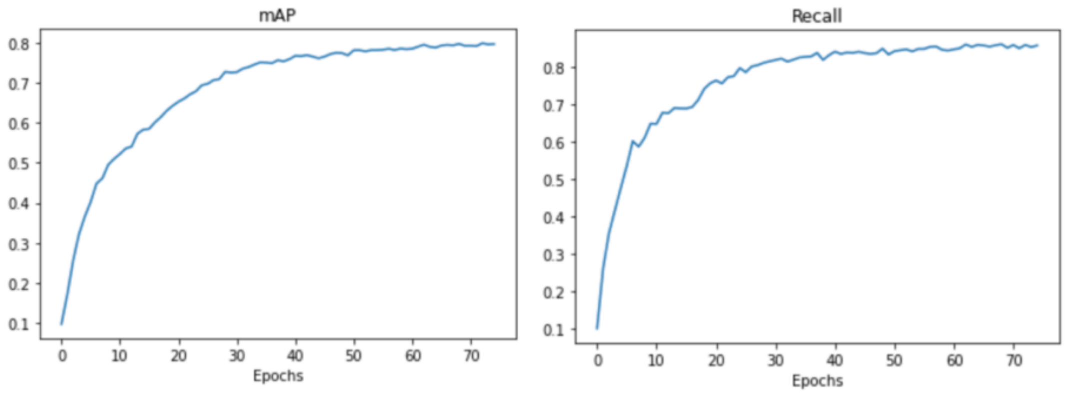


Illustration 3 – Training results for KITTI and YOLOv3

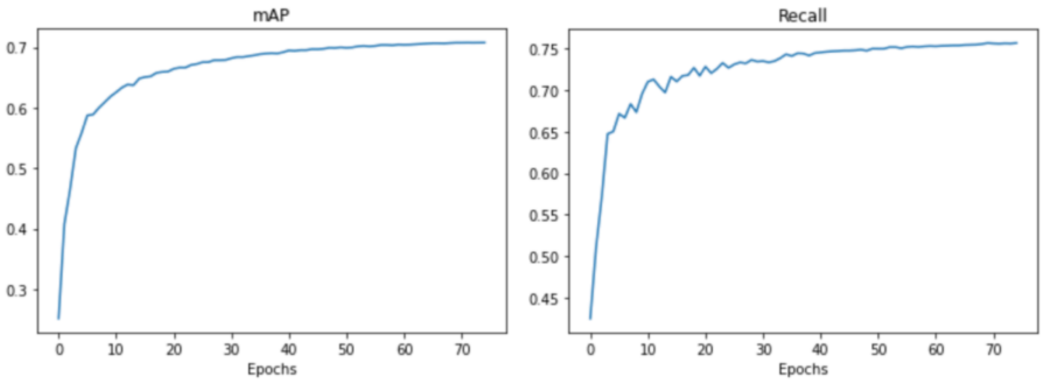


Illustration 4 – Training results for CrowdHuman in YOLOv3

Although these results seem that they are not much worse than those obtained with YOLOv5, in processing time they are, since YOLOv5 is capable of processing images 20 FPS faster. In addition, it is also better in detections, especially with respect to KITTI, as seen below, KITTI is not capable of generalizing well and produces results with much to be desired in inference.

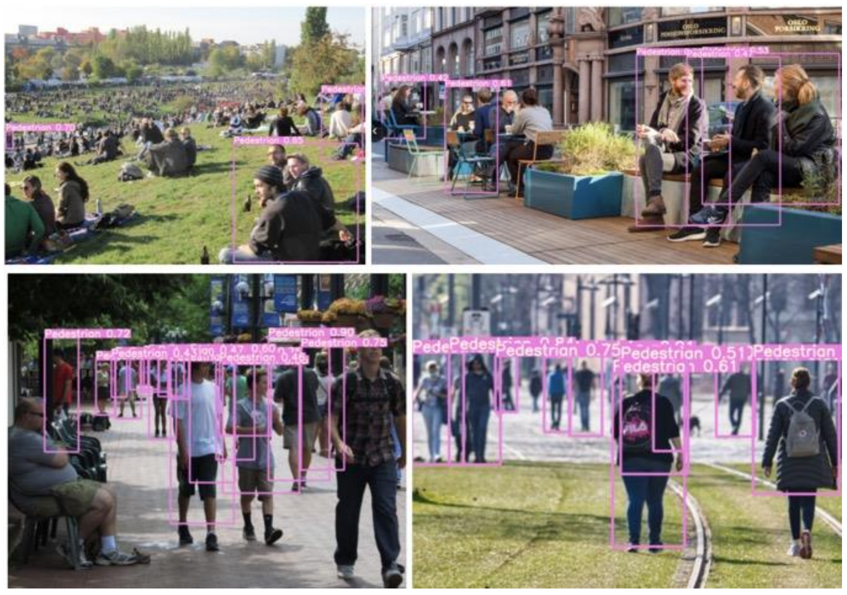


Illustration 5 – Inference for KITTI in YOLOv3

These KITTI results are due to the fact that it is intended for autonomous driving tasks, and it is not able to generalize to scenarios where the perspective is different.

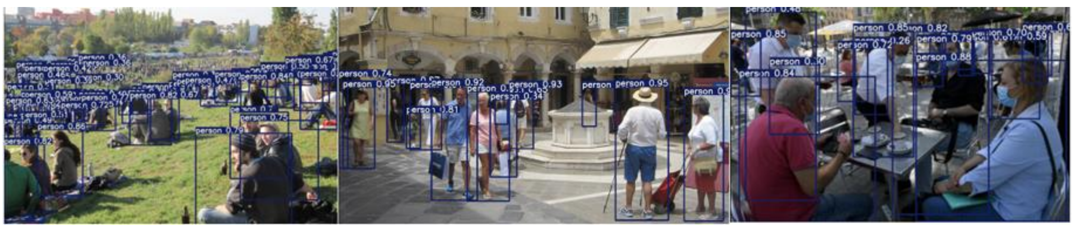


Illustration 6 – Inference for CrowdHuman in YOLOv5 small

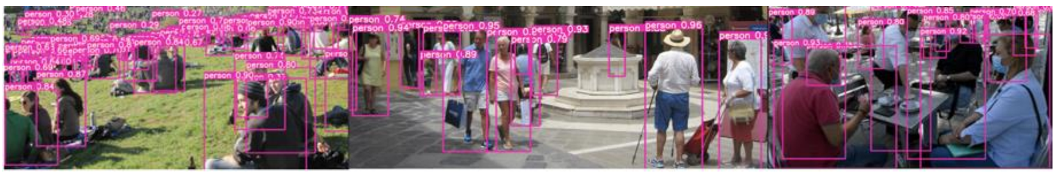


Illustration 7 – Inference for CrowdHuman in YOLOv5 medium

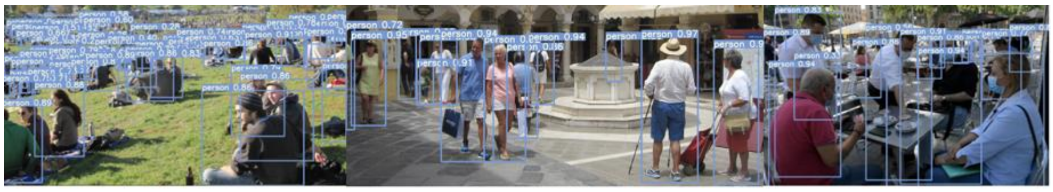


Illustration 8 – Inference for CrowdHuman in YOLOv5 large

As can be seen, YOLOv5, specially trained in CrowdHuman, is much more robust and flexible than both YOLOv3 and KITTI. Next, results will be shown in COCO so that the operation of all the datasets can be verified.

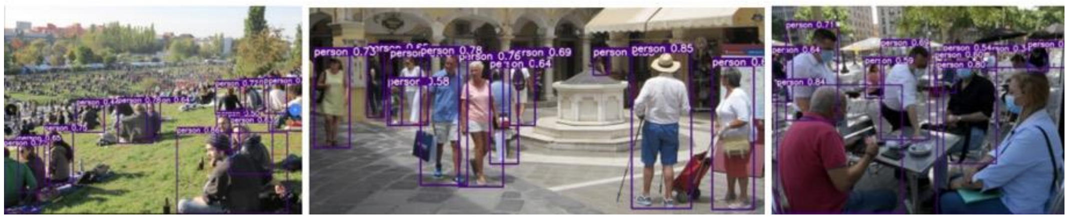


Illustration 9 – Inference for COCO in YOLOv5 medium

It looks like as for COCO, CrowdHuman provides better results detecting people in long distance camera shots. The YOLOv3 results are shown below, where you can see that the model is clearly inferior to YOLOv5. Despite this, it performs better than any KITTI-trained model.

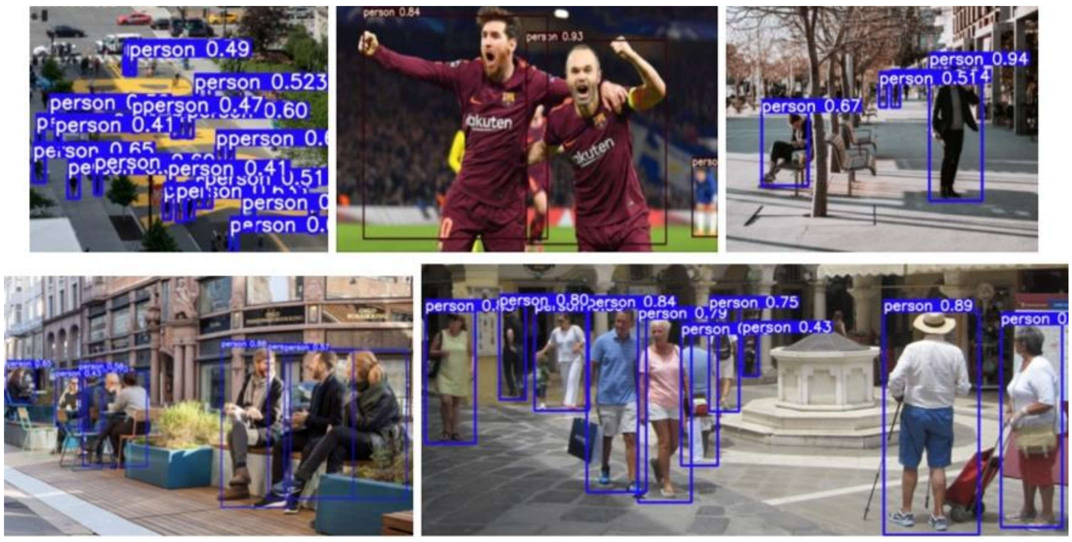
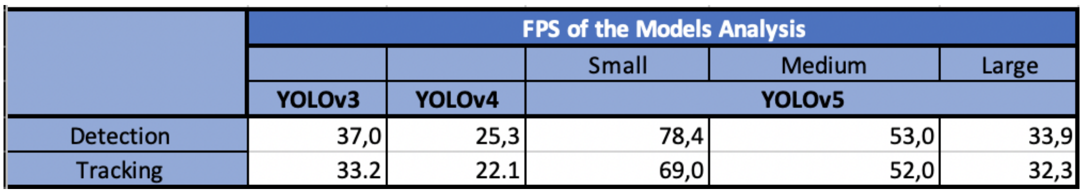


Illustration 10 – Inference for COCO in YOLOv3

A complete evaluation of all the models and their performance based on detections and processing speed can be found here.

- **Tracking**:

DeepSORT is the state-of-the-art model for real-time tracking and the results obtained by using the old detector on the tracker are consistent with that. Here is a comparison of the different detection models that have been tested with the tracking model in terms of FPS and detector and tracker inference:



	FPS of the Models Analysis				
			Small	Medium	Large
	YOLOv3	YOLOv4	YOLOv5		
Detection	37,0	25,3	78,4	53,0	33,9
Tracking	33.2	22.1	69,0	52,0	32,3

Illustration 11 – Model comparison in terms of FPS

As commented before, for this stage transfer learning was used taking as the pretrained model the authors model and weights, but only the DeepSORT ones. To use DeepSORT, a trained detector is needed and also a trained tracker. For this stage the previously trained detectors were used and fed into the pretrained tracker. The authors [3] trained the model on the MOT challenge, a challenge consisting on tracking multiple objects, especially people in video. Since that is very similar to this task, using that pretrained model has been very accurate and useful.

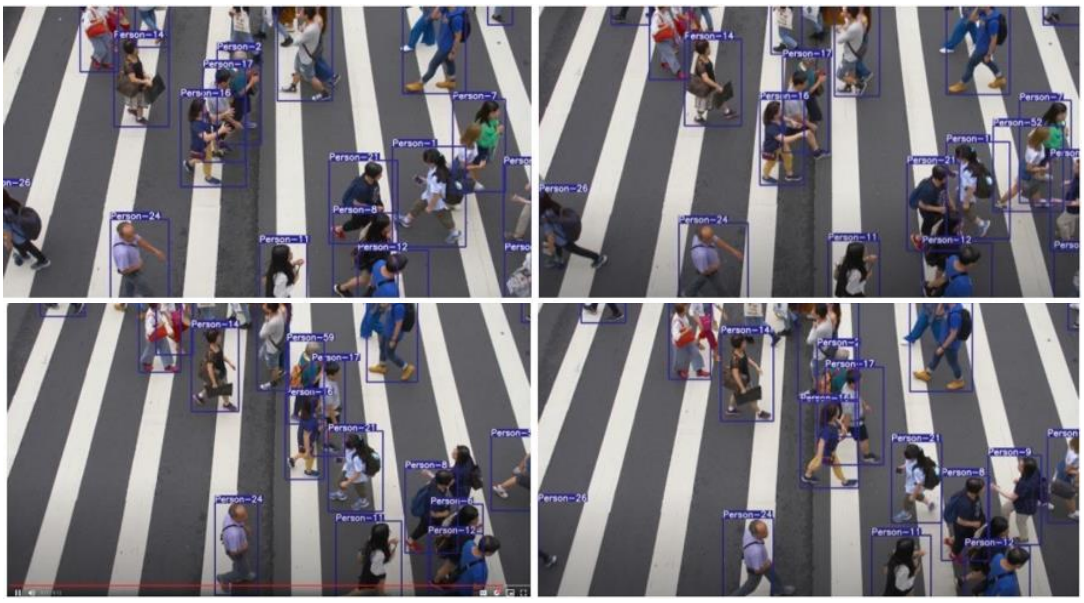


Illustration 12 – Inference of Detecor and Tracker in a video - 1

In the image it cannot be verified, but with this algorithm the system is capable of not losing detections due to occlusions in addition to working within the limits of real time. It is more clearly seen in the following sequence, where the detection of the man in the foreground is not lost despite a cyclist passing in front.

Illustration 13 – Inference of Detecor and Tracker in a video – 2

In order to check all the inference videos for all the models into the tracker, it can be done here.

- **Distance estimation**:

After trying different approaches to the distance estimation phase, including CNN networks directly in images, translating the bounding box from color to depth and DNN images, the best-performing method is construction, from a set of synthetic data built from a scenario simulator with people, another dataset in CSV format with coordinates of the bounding box of those people (using the previous detectors) and the pixel values of the center of those bounding boxes, which allows us to use some in-depth information.

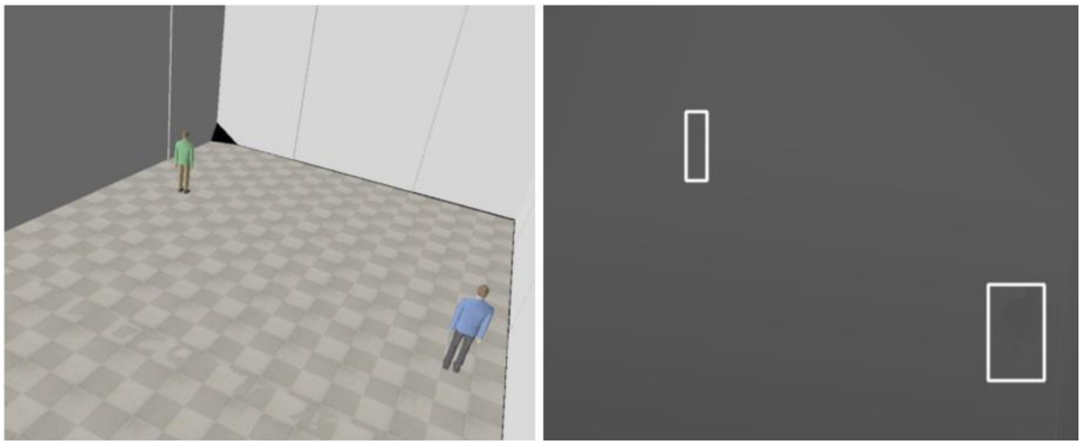


Illustration 14 – Generating the synthethic dataset

The synthetic dataset consists of image pairs like Illustration 13, although the model is only given the one on the right. The images like the one on the left are all passed through the detector and moved to exactly the same positions in the depth images, where the shapes are not differentiated by human eyes, but the computer is capable of doing so, thus obtaining its information. deep. This dataset is used as input to the CNN model, obtaining the following results (with distances in meters).

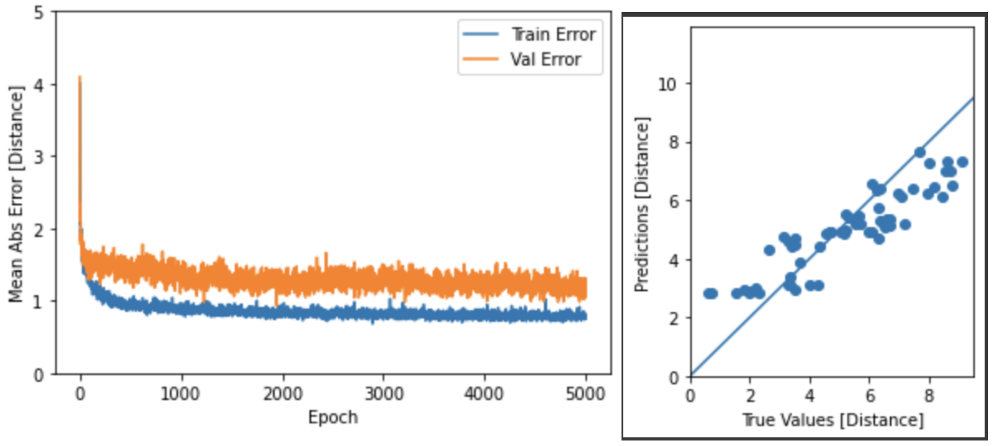
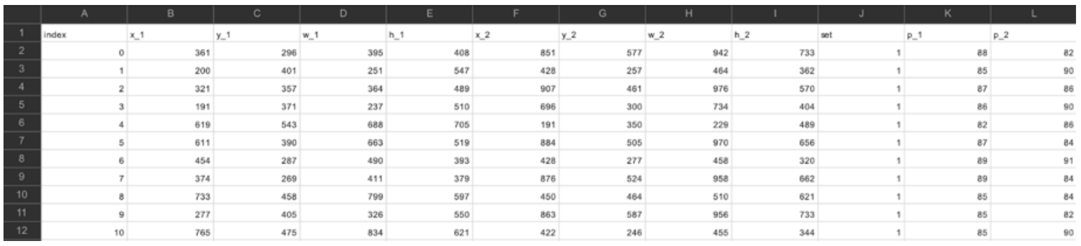


Illustration 15 – Results of the CNN based model

The CNNs model has as MAE a value of 0.93 m in training and between 1.2 m and 1.6 m in test. Although it is not a bad result considering the simplicity of both the dataset and the model, I believe that under these conditions it can be improved. Still, doing almost the scatter plot, it is seen how the predictions are not far from the real values.

Even so, a DNN model has been tested and has provided better results. This model uses a dataset that has the information on the size and central coordinates of the detections (that part of the process does not change), the depth values of the central point of the detections, such as depth information and the distance as a target, thus being distributed (the distance does not appear since it has been conveniently separated for the execution of the model).



	A	B	C	D	E	F	G	H	I	J	K	L
1	index	x_1	y_1	w_1	h_1	x_2	y_2	w_2	h_2	set	p_1	p_2
2	0	361	296	395	408	851	577	942	733	1	88	82
3	1	200	401	251	547	428	257	464	362	1	85	90
4	2	321	357	364	489	907	461	976	570	1	87	86
5	3	191	371	237	510	696	300	734	404	1	86	90
6	4	619	543	688	705	191	350	229	489	1	82	86
7	5	611	390	663	519	884	505	970	656	1	87	84
8	6	454	287	490	393	428	277	458	320	1	89	91
9	7	374	269	411	379	876	524	958	662	1	89	84
10	8	733	458	799	597	450	464	510	621	1	85	84
11	9	277	405	326	550	863	587	956	733	1	85	82
12	10	765	475	834	621	422	246	455	344	1	85	90

Illustration 16 – CSV Dataset

This data set is the input to the model and the labels are the distances measured in the simulator. The results, while not perfect for many reasons explained in detail throughout the document, are good enough for the proposed task. Above all, they are much better than the previous model but both could serve, in the future and after the relevant improvements, for this task.

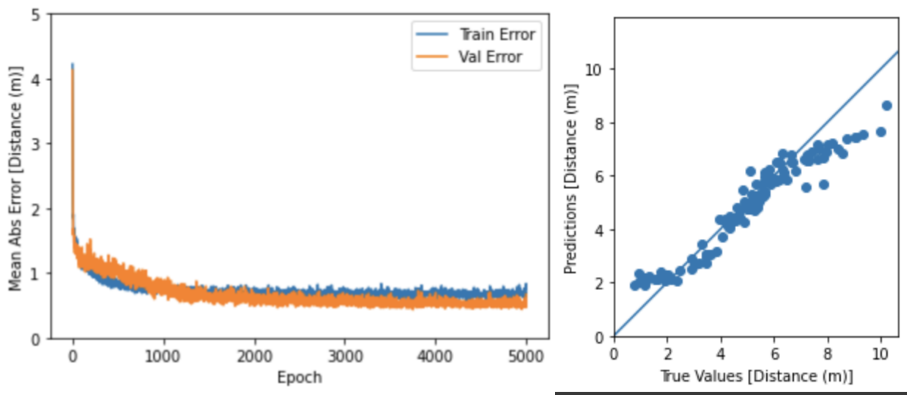


Illustration 17 – Results of the DNNs based dataset

Compared to the previous model, this model has much less bias, especially in short and medium distances, and less variability, following the real values of the distances more precisely.

5. Conclusions

Considering the results presented in the last section, the monitorization of the interpersonal distance using deep learning while having a processing speed within the boundaries of real time (30 FPS) is possible.

After the detection phase, with the YOLOv5 model trained the model is very accurate, it has some difficulties detecting people from very long distances and has some trouble with occlusions. However, it is very fast. For now, long distance detection is not a problem of much concern, as long as short distance detection is good, which it is. The occlusions, however, are a problem and we need to deal with them. That is why we incorporate a tracking mechanism. To try to avoid occlusions DeepSORT was implemented as the tracker. It is based on Kalman Filters but incorporates a Deep Convolutional Network that works as an appearance descriptor between frames. Comparing the output of it to determine if a detection is very similar to one in the previous frame, we can “recover” lost detections some frames later.

Looking at the results and inference time of the YOLOv5 model + DeepSORT model, the outcome is very satisfactory. Occlusions are corrected in a lot of cases. Of course, this is not an exact science and between frames, as explained, one detection can occupy the space of another and be very similar. Unfortunately, some of those occlusions are not corrected, but that is rare and overall performance is very good. In terms of FPS, the processing speed is obviously reduced since the images pass through another model, but the FPS of the models selected are still above 50 FPS, a speed well above the limits stablished. Since the last stage is a very simple and light model, the processing speed is not expected to be reduced so, the real-time objective is reached with a considerable margin.

To estimate the distance, we need some kind of distance annotations to train the model against them and check accuracy etc. There are no datasets like this so, in order to reach a deep learning solution for the problem, a synthetic dataset was created. This is very archaic, and it is not a state-of-the-art dataset but, as shown in the results, is good enough for now. The correct thing to do would be to use a photorealistic simulator. While using a very simple dataset, a very simple DNN model reached a MAE of only 0.6 m, that is not excellent but is good enough, to check that estimating the distance is possible using deep learning.

With all these stages producing good results, estimating the distance between individuals is possible using a deep learning approach and with real time processing speed. Of course, there is much work ahead, but this could be a first step towards a much more general and precise solution in the future.

6. References

[1] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[1] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[2] N. Wojke, A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.

[3] G. Ning et al., "Spatially supervised recurrent convolutional neural networks for visual object tracking," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050867.

[4] Deepak Biria´s Project: https://blog.usejournal.com/social-distancing-ai-using-python-deep-learning-c26b20c9aa4c

[5] YOLOv5 🚀 and Vision AI ⭐ (ultralytics.com)

[6] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement" 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1804.02767.

[7] Inference of all the models tried: **Annex II**

Monitorización de la Distancia Interpersonal mediante Aprendizaje Profundo y Visión por Ordenador

Autor: López de Toledo Soler, Rodrigo
Directores: Boal Martín-Larrauri, Jaime; López López, Álvaro Jesús
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En estos tiempos extraños vividos el último año y medio, mantener una distancia de seguridad entre las personas para evitar que el Covid-19 se propague, se ha convertido en una de las medidas más efectivas para controlar la pandemia. El objetivo de este proyecto es determinar si es posible monitorizar esa distancia utilizando un enfoque de aprendizaje profundo que brinde independencia de la posición de la cámara y pueda ofrecerlo en tiempo real. Después de entrenar diferentes modelos para la detección, seguimiento y estimación de distancia entre individuos, los resultados muestran que, con alguna mejora, este objetivo se puede lograr dentro de los límites del video en tiempo real.

Palabras clave: Aprendizaje profundo, visión por ordenador, detección, tracking, estimación de la distancia

1. Introducción

La visión por computadora es un área de estudio que se identifica como un subcampo de la inteligencia artificial y trabaja en paralelo con el aprendizaje automático. Hace uso de métodos especializados y algoritmos generales de Machine Learning. El objetivo de la visión por computadora es la comprensión del contenido de las imágenes digitales y, por lo tanto, los videos, para imitar la capacidad de la visión humana. Para los humanos, entender una imagen es muy fácil, es natural y sin esfuerzo, pero para una computadora es sumamente complicado porque es muy difícil entender los cambios de iluminación, posición, orientación y expresión que vemos de manera tan natural. A principios de la década de 1970, la visión por computadora era solo un punto ambicioso en la agenda de dar a los robots una inteligencia humana. En las décadas de 1980 y 1990, el enfoque cambió, los investigadores centraron sus esfuerzos en técnicas matemáticas más complejas para realizar análisis de imágenes.

Es importante comprender que el procesamiento de imágenes que utiliza técnicas comunes como el ajuste de la imagen y el contraste, la ecualización por histograma y la binarización, no es visión por computadora. Estas técnicas son muy útiles para aplicarlas a la visión por computadora, pero no son visión por computadora porque no podemos extraer ningún análisis o conocimiento de lo que está sucediendo en la imagen.

Desde el año 2010 (hay que tener en cuenta que la visión por computador nació no hace mucho), han sido varias tareas en el campo que han sido ampliamente investigadas y que han logrado el éxito. Hubo muchas tareas simples que fueron ampliamente investigadas y lograron un gran éxito. Todas estas tareas, desarrolladas haciendo uso de algoritmos de Machine Learning y herramientas de procesamiento de imágenes, fueron detección y reconocimiento de objetos, clasificación de objetos y segmentación de imagenes. Estas sencillas tareas se convierten en herramientas para aplicaciones desarrolladas posteriormente que ya han demostrado su utilidad en el mundo real. Algunos de ellos han sido en tiendas, para un checkout automatizado, imágenes médicas para diagnósticos, vigilancia, biometría y captura de movimiento, siendo el último el foco de este proyecto.

2. Definición del proyecto

El proyecto se basa principalmente en tres fases diferentes que, una vez finalizadas, funcionarán en conjunto y el objetivo final de las tres es proporcionar una estimación precisa de la distancia entre dos individuos en un video. El objetivo es hacer que el sistema funcione en tiempo real, que es la clave para el correcto funcionamiento de este proyecto y la cumplimentación de sus objetivos.

Para la primera parte, la detección, se han probado varios modelos basados en el algoritmo por excelencia en términos de detección en tiempo real que es YOLO [1]. Se han probado las versiones 3, 4 y 5 con diferentes variaciones tanto en hiperparámetros como en tamaño del modelo. Para estas versiones que se han entrenado cada modelo en diferentes datasets y se ha evaluado su rendimiento en términos de detecciones y de velocidad de procesamiento, usando FPS (Frames Por Segundo). Los datasets en cuestión han sido COCO[4], que es el dataset por antonomasia para tareas de detección, KITTI[5], que es un dataset pensado para vehículos autónomos, cuyo objetivo es ver si es capaz de generlizar a otras tareas y CrowdHuman[6], un dataset específicamente preparado para tareas de detección de personas ya que todas sus anotaciones son de humanos. En cuanto a la decisión de los modelos a utilizar, después de hacer un análisis para analizar las diferencias entre modelos basados en RCNN [2] (Region proposal frameworks) y modelos basados en regresión, como es YOLO, se decidió utilizar modelos basados en regresión ya que, a pesar de que la precisión es algo menor, la velocidad al procesar las imágenes es mucho mayor ya que se basan en una sola pasada del algoritmo por cada imagen, en vez de generar muchas posibles regiones y luego compararlas. Debido a eso, la precisión baja, pero la velocidad aumenta, y, teniendo en cuenta que la mayor limitación que hay es el tiempo-real, es mas beneficioso usar un modelo basado en regresión. Además, al poner un algoritmo de tracking detrás, nos servirá como red de seguridad ante posibles errores de detección o pérdidas. Como se menciona anteriormente, el modelo elegido fue YOLO y sus diferentes versiones.

La segunda etapa del proyecto es la de seguimiento. Como se comenta anteriormente, esta parte es una red de seguridad después de la detección. Lo que se busca en esta etapa es que, en caso de que se pierda la detección de una persona en un frame, el sistema sea capaz de reidentificarla si vuelve a aparecer. Esto es muy útil para cuando ocurren oclusiones entre personas que, para una tarea como monitorización de la distancia entre personas es clave poder manejar esas oclusiones y que el sistema pueda, tanto predecir la posición de una detección y que pueda reidentificarla. Para conseguir esta importante característica se ha usado el algoritmo mas reconocido en este ámbito que es DeepSORT [3]. Este algoritmo es una evolución del algoritmo SORT, que se basa en filtros de Kalman para que el sistema tenga memoria y pueda guardar información de detecciones pasadas. Lo que permite este algoritmo es que, una vez hecha una detección, su información se pueda guardar y algunos frames mas tarde recuperarla. Por ejemplo, si

4 https://cocodataset.org/#home

5 http://www.cvlibs.net/datasets/kitti/

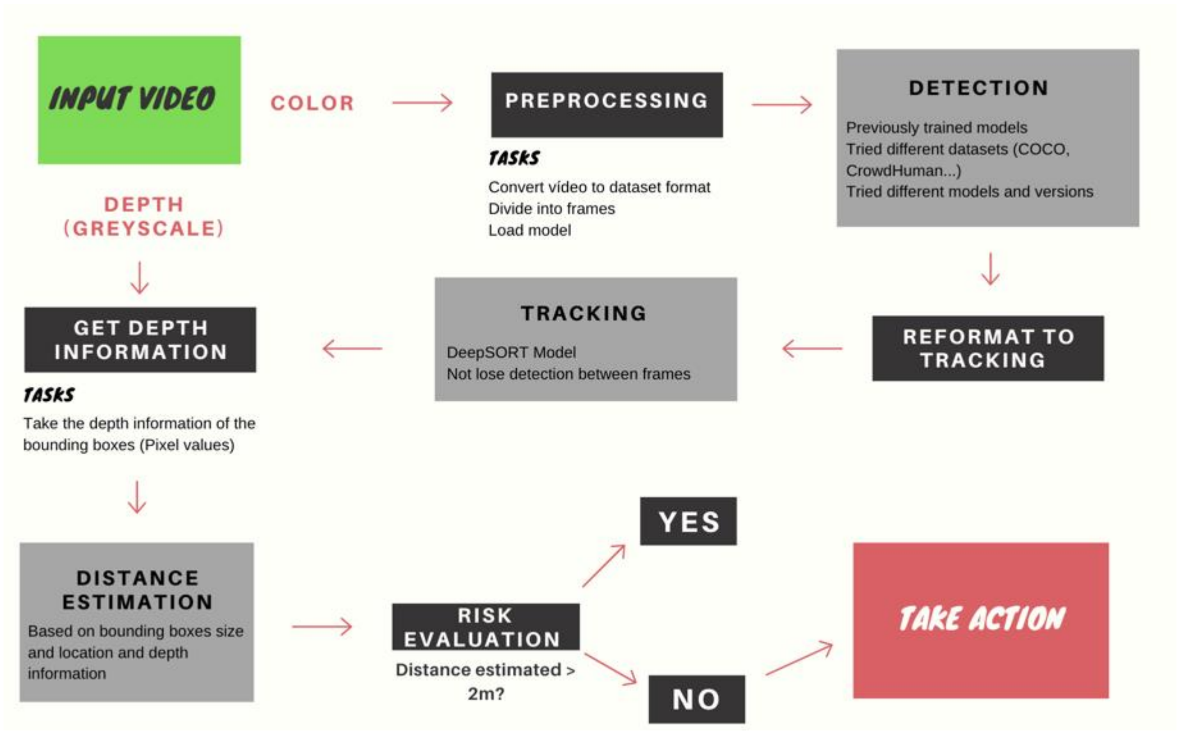
6 https://www.crowdhuman.org/

una persona pasa por detrás de un grupo de personas, el algoritmo guarda la información de velocidad, apariencia y tamaño, entre otras características y, una vez esa persona aparezca de nuevo después de salirse del grupo, el algotimo es capaz de identificarla como la misma detección anterior a la oclusión. Lo que añade DeepSORT es una diferencia fundamental en el extractor de apariencia. Esa parte es la encargada de comparar las detecciones antes y después de la oclusión. SORT [4] hace eso mediante medición de distancias con algoritmos muy conocidos (Húngaro, Distancia coseno…). DeepSORT cambia el enfoque de esta parte y lo que hace es crear una red convolucional que genera un vector de apariencia en base a las *features* extraídas de las imágenes. Ese vector de apariencia es lo que se usa para comparar antes y después de la oclusión. De esta manera, se tiene un sistema mucho mas robusto basado en aprendizaje profundo y que proporciona un seguimiento mas preciso.

Por último, se pasa a la fase de la estimación de la distancia. Para este tipo de tareas se han desarrollado varias soluciones que buscan este objetivo. Sin embargo, ninguna de estas soluciones está basadas en aprendizaje profundo. La mayoría se basan en establecer referencias previas y comparar la posición de cada detección en base a esa referencia previa. Lo mas parecido a aprendizaje profundo es cambiando la perspectiva a vista de pájaro. De esta manera la perspectiva de la cámara es cenital [5] y no hay ningún problema con la profundidad de la imagen. Esto tiene un problema, y es que hay que seleccionar antes de monitorizar a las detecciones sobre las que se quiere medir y monitorizar la distancia. Esto impide que se pueda monitorizar la distancia entre todas las personas de la escena y no se pueden monitorizar a nuevas personas que entren en la escena. Para poder monitorizar todo eso, se implementa un enfoque basado en aprendizaje profundo. Como no existen datasets con anotaciones de distancia, se ha generado un dataset sintético con un simulador donde, iterativamente, se generan dos personas en un escenario, se mide la distancia entre ellas y se extrae el escenario como una imagen, la información de profundidad como otra imagen, pero en blanco y negro, y la medidad de la distancia. Todo eso se junta en un dataset y se usa como entrada al modelo. Se han usado dos modelos en esta fase para poder comparar rendimiento y resultados. Uno de ellos es un modelo basado en redes convolucionales que consta de 4 capas convolucionales con su correspondiente dropout para evitar sobreaprendizaje, con un cuello que consta de una capa completamente conectada de 30 neuronas seguida de otra de 15. Por último, la salida es una única neurona con función de activación lineal cuya saldia es la estimación de la distancia entre las personas de la imagen. El otro modelo es un modelo basado en DNNs. Para poder utilizarlo, se ha puesto la información de todas las detecciones del dataset anterior en forma de CSV, incluyendo las coordinadas y dimensiones de las detecciones, el valor de profundidad de esas detecciones y, como target, la distancia.

3. Descripción del modelo/sistema/herramienta

Como se describe en el cuadro a continuación, el proyecto sigue las tres etapas propuestas con algunos pasos intermedios. El cuadro a continuación muestra la inferencia del proyecto propuesto, pero todos los modelos involucrados (Detección, Seguimiento y Estimación de Distancia), y algunos más, han sido entrenados previamente en diferentes conjuntos de datos y he elegido los de mejores resultados para cada etapa, como se describe en la siguiente sección.



INPUT VIDEO
COLOR
PREPROCESSING
TASKS
Convert vídeo to dataset format
Divide into frames
Load model
DETECTION
Previously trained models
Tried different datasets (COCO, CrowdHuman...)
Tried different models and versions
DEPTH (GREYSCALE)
GET DEPTH INFORMATION
TASKS
Take the depth information of the bounding boxes (Pixel values)
TRACKING
DeepSORT Model
Not lose detection between frames
REFORMAT TO TRACKING
YES
DISTANCE ESTIMATION
Based on bounding boxes size and location and depth information
RISK EVALUATION
Distance estimated > 2m?
NO
TAKE ACTION

Ilustración 1 – Diagrama de Flujo del Sistema

4. Resultados

En esta sección mostraré los mejores resultados que he logrado para cada una de las diferentes fases.

- **Detección**: Para esta fase, el modelo de mejor rendimiento es YOLOv5 [6] entrenado en el conjunto de datos de CrowdHuman. Los resultados del entrenamiento (de modelos de diferentes tamaños) se muestran a continuación:

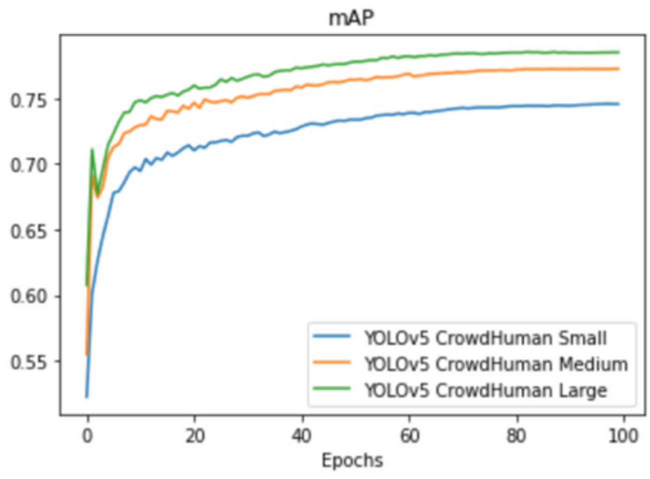


Ilustración 2– Resultados de Entrenamiento de CrowdHuman

Para esta fase, aunque los mejores resultados son con ese modelo, se han probado mas modelos que merece la pena comentar. Se ha probado YOLOv3 [7] con los tres datasets mencionados anteriormente, alcanzando un mAP bastante bueno tanto en KITTI como en CrowdHuman.

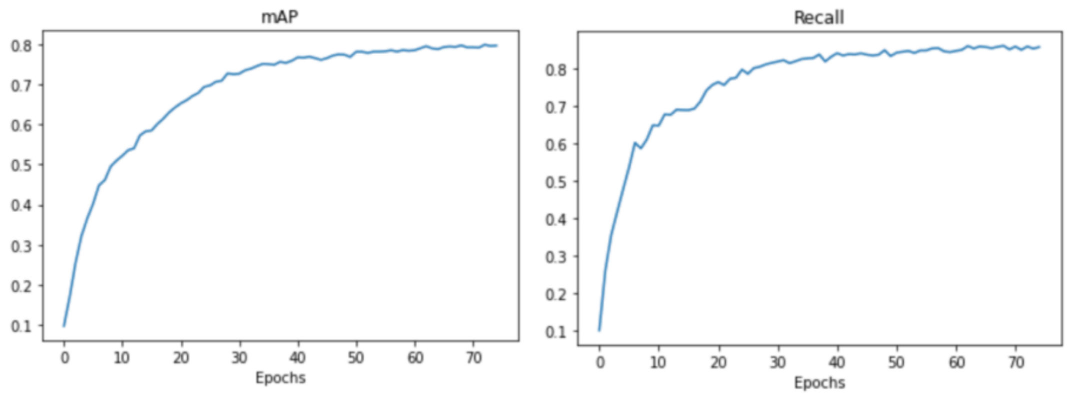


Ilustración 3 – Resultados de Entrenamiento de KITTI en YOLOv3

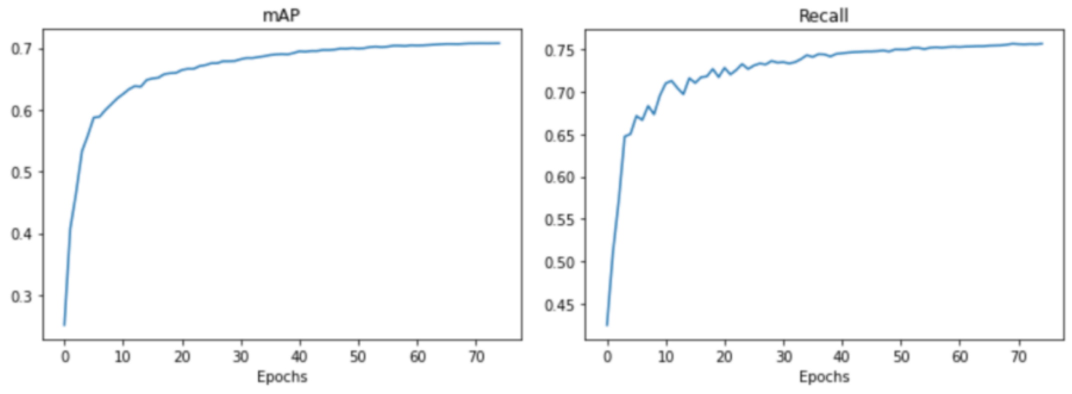


Ilustración 4 – Resultados de Entrenamiento de CrowdHuman en YOLOv3

Aunque estos resultados parecen que no son mucho peores que los que se han obtenido con YOLOv5, en tiempo de procesamiento si lo son ya que YOLOv5 es capaz de procesar las imágenes 20 FPS mas rápido. Además en detecciones también es mejor, sobretodo respecto a KITTI, como se ve a continuación, KITTI no es capaz de generalizar bien y produce resultados con bastante que desear en inferencia.

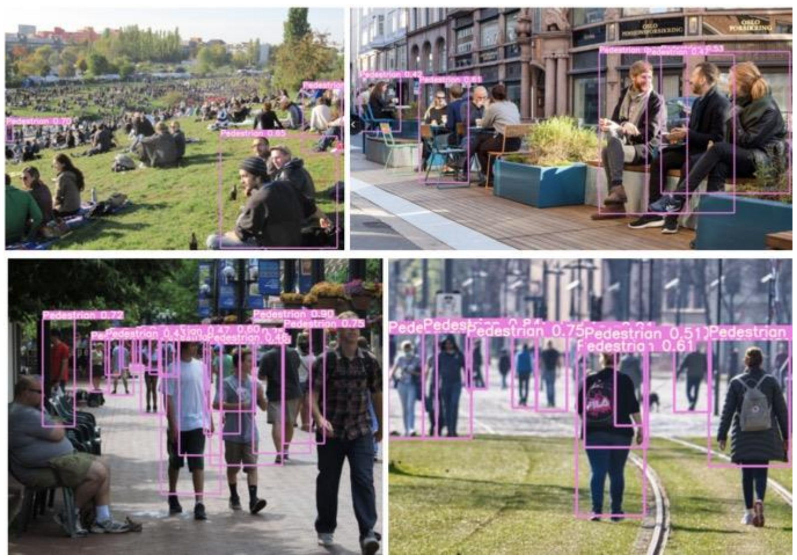


Ilustración 5 – Inferencia de KITTI en YOLOv3

Estos resultados de KITTI se deben a que está pensado para tareas de conducción autónoma, y no es capaz de generalizar a escenarios donde la perspectiva es diferente.

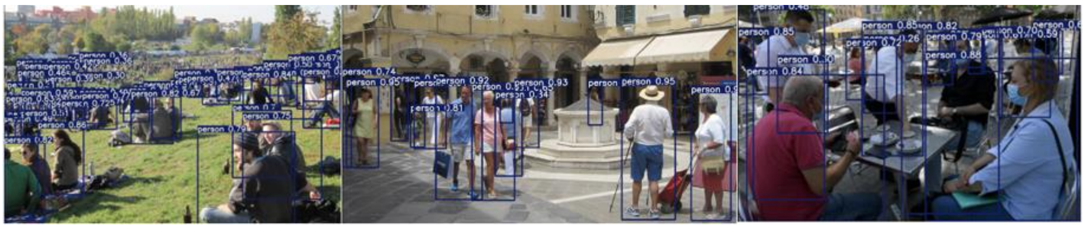


Ilustración 6 – Inferencia de CrowdHuman en YOLOv5 small

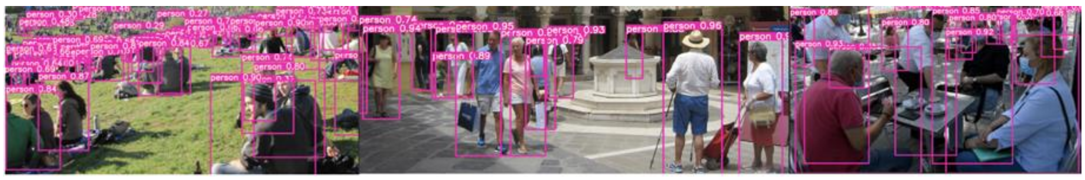


Ilustración 7 – Inferencia de CrowdHuman en YOLOv5 medium

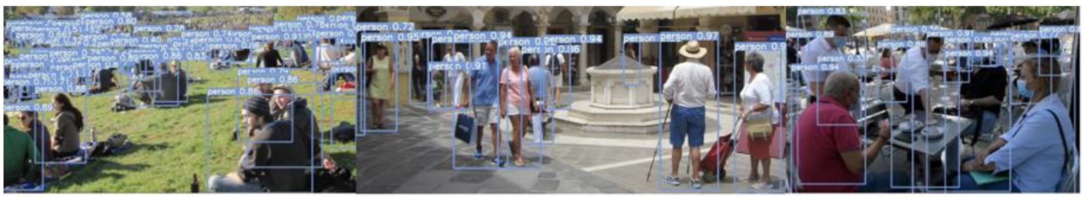


Ilustración 8 – Inferencia de CrowdHuman en YOLOv5 large

Como se aprecia, YOLOv5, especialmente entrenado en CrowdHuman es bastante mas robusto y flexible que tanto YOLOv3 como KITTI. A continuación se mostrarán resultados en COCO para que se pueda comprobar el funcionamiento de todos los datasets.

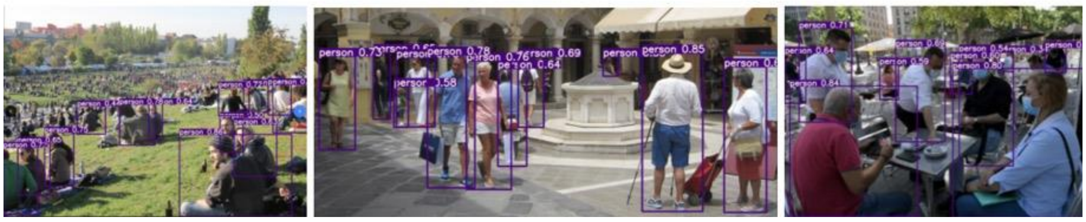


Ilustración 9 – Inferencia de COCO en YOLOv5 medium

Se ve como en cuanto a COCO, CrowdHuman proporciona mejores resultados detectando personas en tiros de cámara de mucha distancia. Los resultados de YOLOv3 se muestran a continuación, donde se puede ver que el modelo es claramente inferior a YOLOv5. A pesar de ello, todos los modelos entrenados con COCO son mejores que los entrenados con KITTI, sobretodo en inferencia.

Ilustración 10 – Inferencia de COCO en YOLOv3

Una completa evaluación de todos los modelos y sus rendimientos en base a detecciones y velocidad de procesamiento se puede encontrar aquí.

- **Seguimiento**: DeepSORT es el modelo de última generación para el seguimiento en tiempo real y los resultados obtenidos mediante el uso del detector anterior en el rastreador son consistentes con eso. A continuación, hay una comparación de los diferentes modelos de detección que se han probado con el modelo de seguimiento en términos de FPS e inferencia del detector y el rastreador:

	FPS of the Models Analysis				
			Small	Medium	Large
	YOLOv3	YOLOv4	YOLOv5		
Detection	37,0	25,3	78,4	53,0	33,9
Tracking	33.2	22.1	69,0	52,0	32,3

Ilustración 11 – Comparacion de modelos en términos de FPS

Como se comentó anteriormente, para esta etapa se utilizó el aprendizaje por transferencia tomando como modelo preentrenado el modelo y los pesos de los autores, pero solo los de DeepSORT. Para utilizar DeepSORT, se necesita un detector capacitado y también un rastreador capacitado. Para esta etapa se utilizaron los detectores previamente entrenados y se introdujeron en el rastreador previamente entrenado. Los autores [3] entrenaron al modelo en el MOT challenge, un desafío que consiste en rastrear múltiples objetos, especialmente personas en video. Dado que es muy similar a esta tarea, usar ese modelo previamente entrenado ha sido muy preciso y útil.

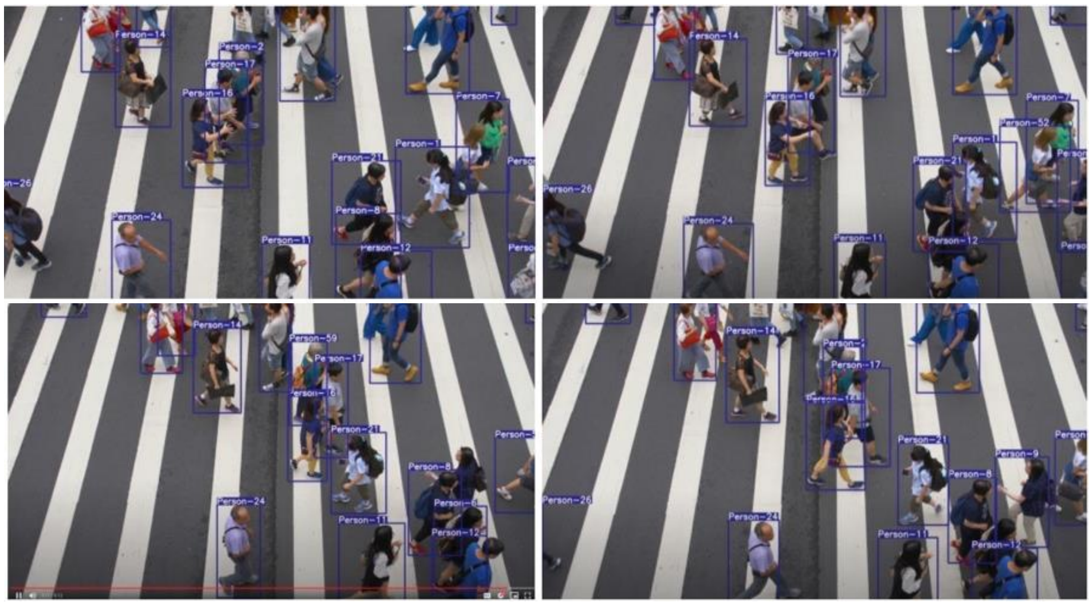


Ilustración 12 – Inferencia del detector y el tracker en un vídeo

En la imagen no se puede comprobar, pero con este algoritmo el sistema es capaz de no perder detecciones por oclusiones ademaás de trabajar dentro de los límites del tiempo real. Mas claramente se ve en la secuencia siguiente, donde no se pierde la detección del hombre en primer plano a pesar de que pasa una ciclista por delante.

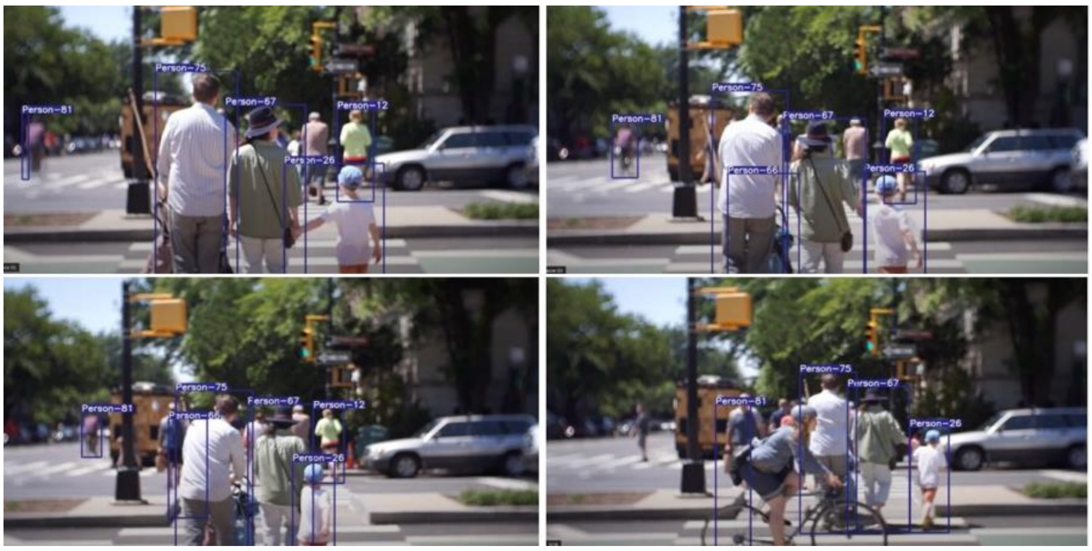


Ilustración 13 – Inferencia del detector y el tracker en un vídeo – 2

Para poder comprobar todos los videos probados en todos los modelos, se facilitan todos los links a donde están guardados los videos e imágenes usados para inferencia aquí.

- **Estimación de distancia**: Después de probar diferentes enfoques para la fase de estimación de distancia, incluidas las redes CNN directamente en imágenes, la traducción del cuadro delimitador de color a imágenes de profundidad y DNN, el método de mejor rendimiento es la construcción, a partir de un conjunto de datos sintéticos construido a partir de un simulador escenarios con personas, otro conjunto de datos en formato CSV con coordenadas del cuadro delimitador de esas personas (usando los detectores anteriores) y los valores de píxel del centro de esos cuadros delimitadores, lo que nos permite usar cierta información de profundidad.

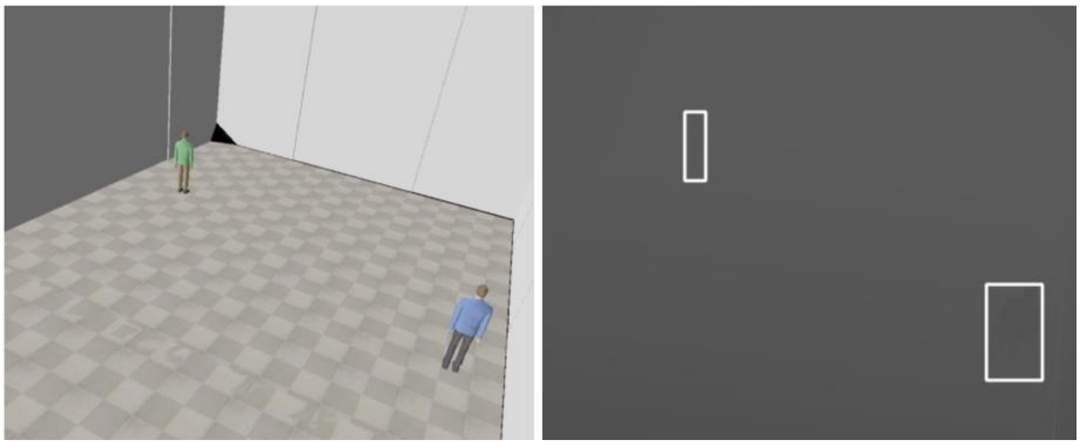


Ilustración 14 – Generación del dataset sintético

El dataset sintético consta de parejas de imágenes como la Ilustración 13, aunque al modelo solo se le da la de la derecha. Las imágenes como la del izquierda se pasan todas por el detector y se trasladan a exactamente las mismas posiciones en las imágenes de profundidad, donde las formas no se diferencian a vista de humano, pero el ordenador si es capaz de hacerlo, obteniendo asi su información de profundidad. Este dataset se usa como entrada al modelo CNN, obteniendo los siguientes resultados (con las distancias en metros).

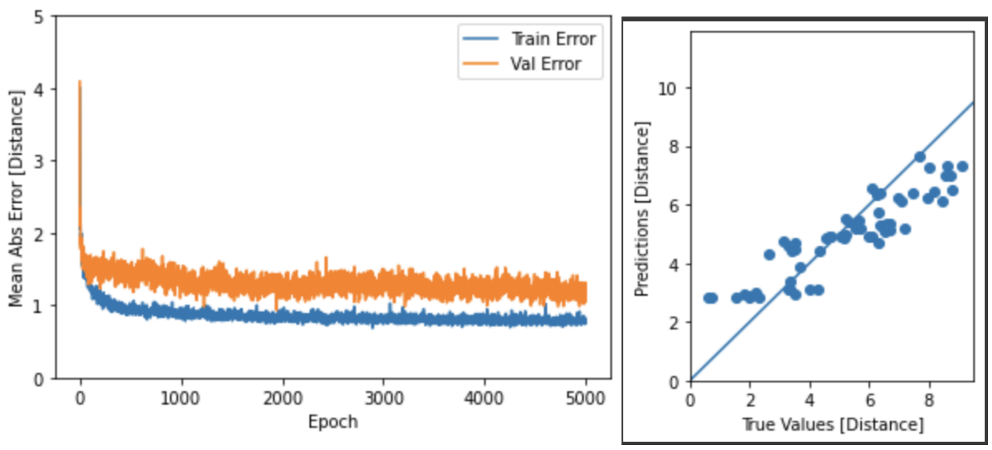
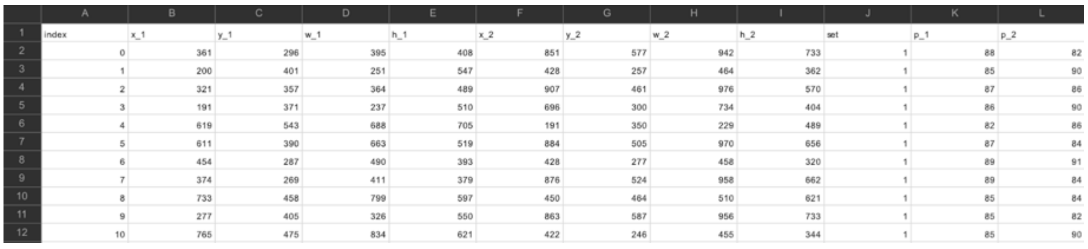


Ilustración 15 – Generación del dataset sintético

El modelo de CNNs tiene como MAE un valor de 0.93 m en training y entre 1.2 m y 1.6 m en test. A pesar de que no es un mal resultado teniendo en cuenta la simpleza tanto del dataset como del modelo, creo que bajo estas condiciones se puede mejorar. Aún asi, haciendo casi del scatter plot, se ve como las predicciones no se alejan mucho de los valores reales.

Aun así se ha probado un modelo de DNN que ha proporcionado resultados mejores. Este modelo usa un dataset que tiene la información de tamaño y coordenadas centrales de las detecciones (esa parte del proceso no cambia), los valores de profundidad del punto central de las detecciones, como información de profundidad y la distancia como target, quedando distribuido así (la distancia no sale ya que se ha separado convenientemente para la ejecución del modelo).



	A	B	C	D	E	F	G	H	I	J	K	L
1	index	x_1	y_1	w_1	h_1	x_2	y_2	w_2	h_2	set	p_1	p_2
2	0	361	296	395	408	851	577	942	733	1	88	82
3	1	200	401	251	547	428	257	464	362	1	85	90
4	2	321	357	364	489	907	461	976	570	1	87	86
5	3	191	371	237	510	696	300	734	404	1	86	90
6	4	619	543	688	705	191	350	229	489	1	82	86
7	5	611	390	663	519	884	505	970	656	1	87	84
8	6	454	287	490	393	428	277	458	320	1	89	91
9	7	374	269	411	379	876	524	958	662	1	89	84
10	8	733	458	799	597	450	464	510	621	1	85	84
11	9	277	405	326	550	863	587	956	733	1	85	82
12	10	765	475	834	621	422	246	455	344	1	85	90

Ilustración 16 – Dataset sintético

Este conjunto de datos es la entrada al modelo y las etiquetas son las distancias medidas en el simulador. Los resultados, aunque no son perfectos por muchas razones explicadas en detalle a lo largo del documento, son lo suficientemente buenos para la tarea propuesta. Sobretodo, son bastante mejores que el modelo anterior, pero ambos podrían servir, en un futuro y tras las mejoras pertinentes, para esta tarea.

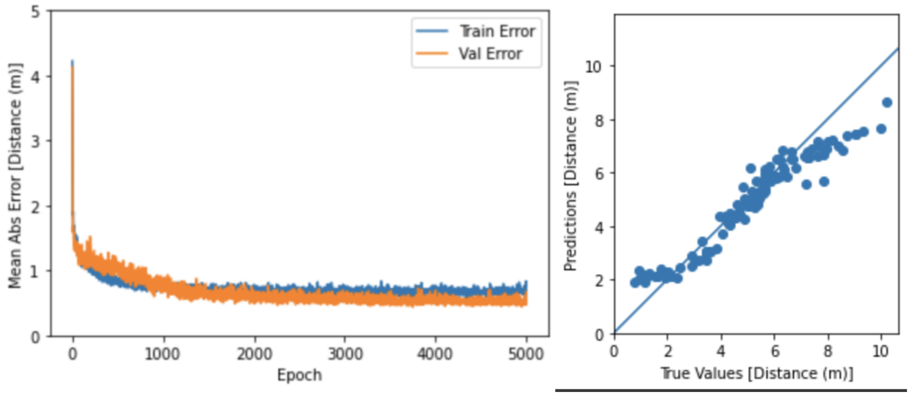


Ilustración 17 – Resultados de training de estimación de la distancia

En comparación con el modelo anterior, este modelo tiene mucho menos sesgo, sobretodo en distancias cortas y medias, y menos variabilidad, siguiendo de una manera mas precisa los valores reales de las distancias.

5. Conclusiones

Teniendo en cuenta los resultados presentados en la última sección, es posible el monitoreo de la distancia interpersonal utilizando deep learning mientras se tiene una velocidad de procesamiento dentro de los límites del tiempo real (30 FPS).

Después de la fase de detección, con el modelo YOLOv5 entrenado, el modelo es muy preciso, tiene algunas dificultades para detectar personas desde distancias muy largas y tiene algunos problemas con las oclusiones. Sin embargo, es muy rápido. Por ahora, la detección de larga distancia no es un problema de mucha preocupación, siempre que la detección de corta distancia sea buena, que es. Las oclusiones, sin embargo, son un problema y debemos ocuparnos de ellas. Por eso incorporamos un mecanismo de seguimiento. Para tratar de evitar oclusiones, se implementó DeepSORT como rastreador. Se basa en los filtros de Kalman, pero incorpora una red convolucional profunda que funciona como un descriptor de apariencia entre fotogramas. Comparando la salida del mismo para determinar si una detección es muy similar a una en el cuadro anterior, podemos "recuperar" las detecciones perdidas algunos cuadros más tarde.

Mirando los resultados y el tiempo de inferencia del modelo YOLOv5 + modelo DeepSORT, el resultado es muy satisfactorio. Las oclusiones se corrigen en muchos casos. Por supuesto, esto no es una ciencia exacta y entre fotogramas, como se explicó, una detección puede ocupar el espacio de otra y ser muy similar. Desafortunadamente, algunas de esas oclusiones no se corrigen, pero eso es raro y el rendimiento general es muy bueno. En cuanto a FPS, la velocidad de procesamiento es obviamente reducida ya que las imágenes pasan por otro modelo, pero los FPS de los modelos seleccionados siguen estando por encima de los 50 FPS, una velocidad muy por encima de los límites establecidos. Dado que la última etapa es un modelo muy simple y ligero, no se espera que la velocidad de procesamiento se reduzca, por lo que el objetivo en tiempo real se alcanza con un margen considerable.

Para estimar la distancia, necesitamos algún tipo de anotaciones de distancia para entrenar el modelo contra ellas y verificar la precisión, etc. No hay conjuntos de datos como este, por lo que, para llegar a una solución de aprendizaje profundo para el problema, se creó un conjunto de datos sintéticos. Esto es muy arcaico y no es un conjunto de datos de última generación, pero, como se muestra en los resultados, es lo suficientemente bueno por ahora. Lo correcto sería utilizar un simulador fotorrealista. Al usar un conjunto de datos muy simple, un modelo DNN muy simple alcanzó un MAE de solo 0.6 m, que no es excelente, pero es lo suficientemente bueno, para verificar que la estimación de la distancia es posible usando el aprendizaje profundo.

Con todas estas etapas produciendo buenos resultados, es posible estimar la distancia entre individuos utilizando un enfoque de aprendizaje profundo y con velocidad de procesamiento en tiempo real. Por supuesto, queda mucho trabajo por delante, pero este podría ser un primer paso hacia una solución mucho más general y precisa en el futuro.

6. Referencias

[1] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[2] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[3] N. Wojke, A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.

[4] G. Ning et al., "Spatially supervised recurrent convolutional neural networks for visual object tracking," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050867.

[5] Deepak Biria´s Project: https://blog.usejournal.com/social-distancing-ai-using-python-deep-learning-c26b20c9aa4c

[6] YOLOv5 🚀 and Vision AI ⭐ (ultralytics.com)

[7] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement" 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1804.02767.

[8] Inference of all the models tried: **Annex II**

Index

List of figures

List of tables

CHAPTER 1. INTRODUCTION

Since artificial intelligence and machine learning came to our lives, quite recently in fact, several fields of investigation and research branched out from it. As happened with this new "science" in other fields like, for example finance, the use of images and videos for further analysis and development was one of the main fields of interest, and therefore, computer vision was born.

Computer vision is an area of study that is identified as a subfield of artificial intelligence and works in parallel with Machine Learning. It makes use of specialized methods and general algorithms of Machine Learning. The goal of computer vision is the understanding of the content of the digital images and therefore, videos, to mimic the ability of the human vision. For us humans, understanding an image is very easy, it comes natural and effortless, but for a computer it is extremely complicated because it is very difficult to understand the changes in lighting, position, orientation and expression that we see so natural. In the early 1970s, computer vision was only an ambitious point in the agenda of giving robots a human intelligence. In the 1980s and 1990s, the focus shifted, researchers centered their efforts on more complex mathematical techniques to perform image analysis.
It is important to understand that image processing using common techniques such as image and contrast adjustment, histogram equalization and binarization is not computer vision. These techniques are very useful to apply to computer vision, but they are not computer vision because we cannot extract any analysis or knowledge from what is happening in the image.

Since the year 2010 (bear in mind that computer vision was born not so long ago), there have been several tasks in the field that have been extensively researched and that have achieved success. There were a lot of simple tasks that were widely researched and achieved great success. All these tasks, developed by making use of Machine Learning algorithms and image processing tools, were object detection and recognition, object classification and object segmentation. These simple tasks become tools for applications developed later and proven useful in the real-world. Some of them have been retail, for automated checkout, medical imaging, surveillance, biometrics and motion capture, being the last one the focus of this thesis.

All these tasks and applications can be used for the same activity, for example, for surveillance applications, motion capture is essential to detect possible security breaches and biometrics to prevent them. For our objective, which is monitoring interpersonal distance, we need to make use of several of these methods. We need to use motion capture techniques to analyze the pattern and direction a person walks in, object detection and object recognition to check if they are complying with the actual regulations of health and security and also image segmentation to measure that distance.

1.1 MOTIVATION BEHIND THIS PROJECT

In these weird times we are living in, when without knowing we may or may not be carrying a virus that has expanded all over the world and has taken an unbelievable amount of lives, following the health and security instructions recommended by the authorities become crucial. Since we are in 2020 and technology is now our way of understanding everything, to ensure all these measures are being obeyed by the population, we need a way of making sure they are, and, since the number of people controlling is very small in comparison to the people that needs to be controlled, using all the "unofficial" controlling methods such as cameras located all over the cities is almost mandatory. Using machine learning and the recent development in computer vision this can possibly be handled and hopefully be useful not just for now, but for other future situations we may face. But before diving in all the technicalities, I find the need of putting some context to it.

1.2 ETHICAL & SOCIETAL IMPLICATIONS OF DEEP LEARNING AND COMPUTER VISION

Before that, addressing the concern around privacy and security in AI and computer vision specifically is necessary so the primary issues with ethics and computer vision societal impact will be described next. It is not until the last decade that computer vision has reached the everyday life of humans. You can see that in multiple applications, security and autonomous vehicles, as mentioned before, applications like Google Maps, quality assurance applications, health imagery and even motion capturing video games. With this intrusion, an important question arises, how do engineers, researchers and companies protect our privacy and how deep computer vision can get into our normal life?

From the researchers and engineers' perspective, as a fellow engineer, the privacy and security of us all start with us. We need to be aware of the ethical ramifications of our work since the beginning. It would be a shame to achieve great results in something that won´t see the light of day because we didn't take care and comply with the privacy regulations from the get-go [1].

In general terms, there are five main concerns regarding ethics in computer vision which are [2]: Espionage, Identity Theft, Malicious Attacks, Discrimination and Misinformation. In computer vision terms these commonly known terms may have a different meaning from what you may think. Espionage refers to the collection of data on individuals without their consent and it is present in face recognition, character recognition and action and motion applications. From an ethical point of view, it is a serious breach of an individual´s privacy, for example when distributed within social networks. Additionally, when used in a social network environment, tags are chosen freely by users that may reveal user preferences, ties and activities. Identity theft is stealing another individual identity for malicious purposes. Computer vision image classification can accurately identify sensitive information. Even though this is a punishable crime and obviously a privacy breach, this only applies tangentially to our case. Targets for these attacks usually are credit cards, passports and other identification similar to them, which are not the subject of our computer vision application, so an identity theft breach is highly unlikely to happen in this case. The only possible link to our project is when combined with malicious attacks. As we are probably going to store data and identifications with our application, we cannot let those identifications see the light of day. In case a malicious attack targets us, then we may lose private data and maybe allow identity theft to happen. This would be a disaster and we need to monitor very closely any possibility of a malicious attack. When face detection techniques face skin detection, skin coloration and facial features related to different ethnicities can be used to discriminate groups of people. In some cases, like surveillance, when using deep learning techniques to predict, for example, a home invasion, discrimination issues can appear when training the model to identify targets. This is called racial profiling and it is quite a popular subject nowadays. Since we try to mimic human behavior with artificial intelligence, if us humans often fall in the mistake of racially profiling an individual or group of individuals, it is logical that machine learning algorithms also do that. In our case, it is unlikely that we face this issue, because we want to identify if someone wears masks, maintain a sufficient distance with other people etc. It is very unlikely that we cluster group of people for that. Finally, misinformation is transmitting media and data in a way that is deceptive to individuals. In our case, in case it develops to this stage, which is not the main objective for now, it will be necessary to address this issue, since identifying a mask wearer as a non-mask wearer and applying the corresponding punishment that authorities give will become a problem and our application needs to avoid these false positives identifications.

In conclusion, for our project and its corresponding scope, we need to watch as closely as possible cases of espionage, malicious attacks and misinformation cases. It is important that we tackle these possible cases of this issues as soon as they may appear, whether this is in the early stages, in testing or in the initial planification.

CHAPTER 2. REQUIRED TOOLS

The tools developed for computer vision projects right now are almost endless. One of them is GluonCV, which is an API that can be programmed with several coding language like Python, Julia, Java or R. With the API, multiple models can be deployed like Faster CNNs or ResNet models. GluonCV is based in Apache and implemented in Python for all available systems [3][4][5].

OpenCV is another tool similar to GluonCV which is mainly aimed at real-time computer vision projects. It was developed by Intel on the early 2000s and its early goals was to disseminate vision knowledge. Nowadays the application areas for OpenCV are gesture and facial recognition, human-computer interaction, segmentation and recognition, motion tracking and augmented reality. To support these applications, OpenCV provides a Machine Learning library with several models and Deep Neural Networks architectures. It is written in C++ but is more commonly used in Python.

Our main tools for the idea of this project are not decided yet but, for now, the tools that are favored for that are TensorFlow with the Keras framework, which is already integrated in TensorFlow 2.0, or PyTorch.

TensorFlow is an end-to-end open-source platform developed by Google for machine learning. It has an extensive ecosystem of tools and libraries to build state-of-the-art machine learning applications. World-known brands like Airbnb, Coca-Cola, PayPal, Uber, Twitter and of course Google make use of this environment for their machine learning projects. TensorFlow is based on sensors, which are n-dimensional vectors of any kind of data, and it uses a graph framework, which gives a better portability of the computation.

Why should we use TensorFlow for our project? The flexibility that TensorFlow offers might be a big reason why. TensorFlow offers multiple levels of abstraction so the models that we build, or train are as personally tuned as we want. It is a professional product therefore, we can deploy it almost everywhere from the web to servers or edge devices. The last reason might be the ability to build state-of-the-art models without sacrificing speed or performance. From very simple models to highly complex structures using Keras API can be built with TensorFlow and deployed at a reasonable speed and performance.

As mentioned, TensorFlow can be combined with Keras to build the complex models that we want. What Keras offers is consistent and simple APIs minimizing the user actions required for common use cases. It is built on top of TensorFlow, allowing simple models to scale to large clusters. It is the central part of the TensorFlow ecosystem and covers every step of the machine learning project pipeline, from data management to the tuning of each one of the hyperparameters. It is the tool chosen by most teams in the Kaggle competition of machine and deep learning problems.

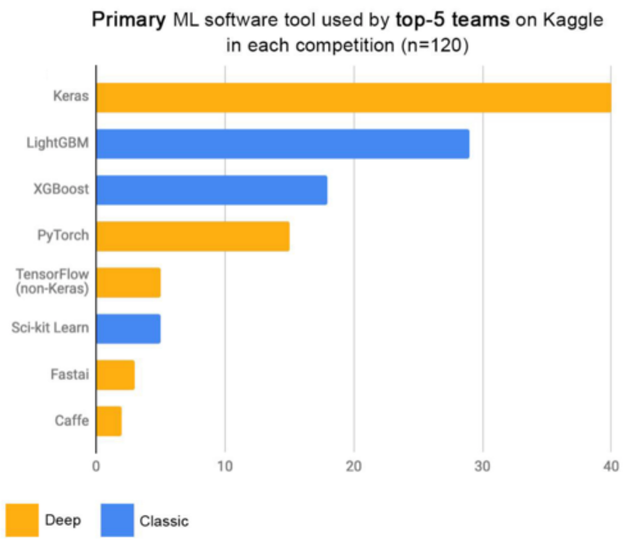


Figure 1. Keras Usage.[7]

As seen in the next picture, it allows researchers and developers to implement their idea in an easy to use, easy to learn environment and visualize the results with the help of TensorBoard, a utility from TensorFlow for visualization. As mentioned before, Keras is integrated in TensorFlow from the second versión, so the autonomous development of Keras is going to stop to develop both tools together.

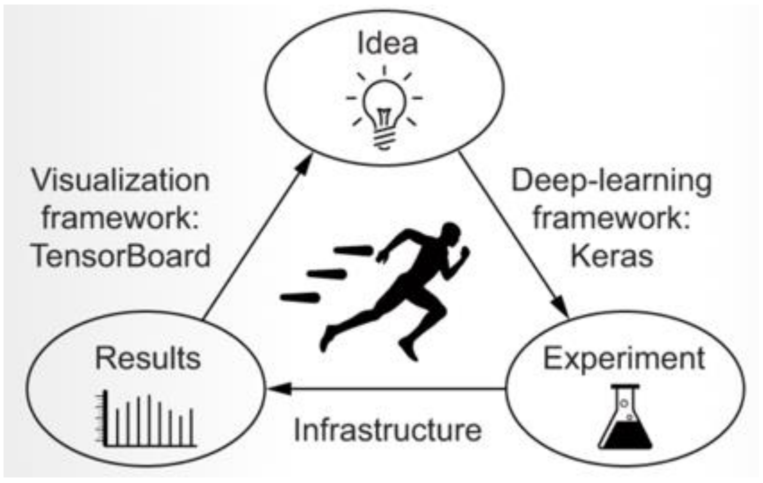

Idea
Deep-learning framework: Keras
Experiment
Infrastructure
Results
Visualization framework: TensorBoard

Figure 2. Keras Approach.[8]

7 https://keras.io/why_keras/

8 https://keras.io

The last tool that was considered was PyTorch, a framework developed by Facebook. PyTorch is another Deep Learning framework very scalable and robust that is used by top companies such as SalesForce. It can be integrated in several cloud platforms with machine learning services such as Google Cloud Platform and AWS. In contrast with TensorFlow, the computation of the graph is not done prior to running the model. It can be done on the go. This dynamic nature of PyTorch works by interpreting the line of code that corresponds to each part of the graph. This cannot be done in TensorFlow. It has fewer features than TensorFlow, but this makes PyTorch an easier to learn tool. It is less supportive than TensorFlow for deployment. For visualization, PyTorch uses CUDA, which is not always available, it depends on the code, the system and the software being used while TensorFlow has TensorBoard integrated into it, where you can visualize the model, you have built. PyTorch is more python friendly which is a characteristic worth monitoring when deciding which framework to use.

CHAPTER 3. STATE OF THE ART

3.1 ADVANCES IN RELATED FIELDS

As mentioned before, closely related to our focus for this project we have security and surveillance and autonomous vehicles, where there has been recent development and they use object detection and motion capture as the base of their research. Some high-level description for some methods and approaches will be mentioned here, before diving into the underlying techniques.

In the field of security, there have been recent achievements to extract actionable insights from data (video) feeds. Privacy is preserved also, that is one of the main concerns with these methods of analysis through video. To be able to prevent or detect in real time these punishable actions computer vision uses 4 main methods which are the following. Object recognition and labeling is one of them. Computer vision detects human presence or objects amazingly well from different angles.

Object detection is basic in computer vision, as well as object classification and image segmentation. All these approaches seem similar, but it is very important to understand their differences, which can be seen in the following images.

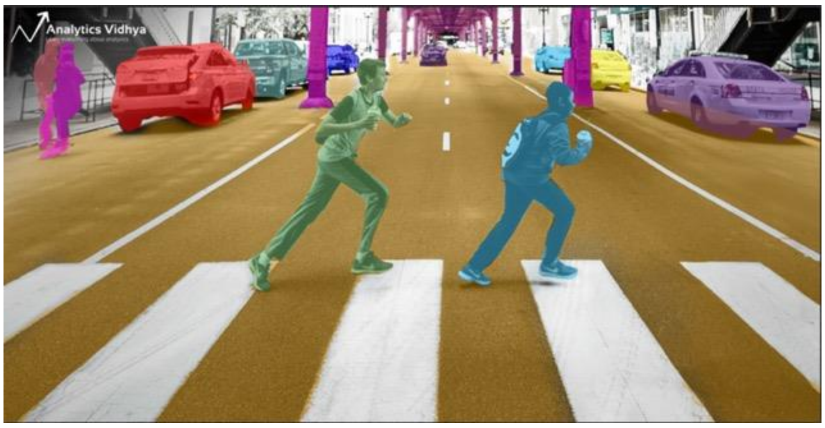


Figure 3. Image Segmentation.[9]

9 https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/

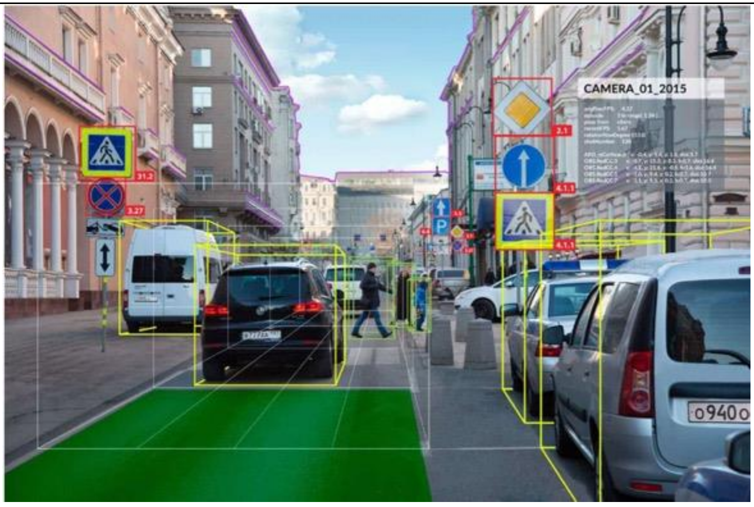


Figure 4. Object Detection.[10]

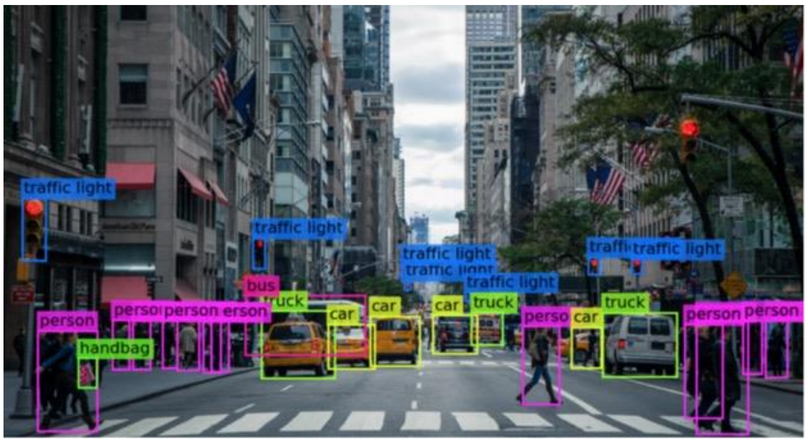


Figure 5. Object Tagging.[11]

As seen in the three previous images, those approaches are quite similar, but they are not the same. It can be seen as a workflow, in fact, image segmentation serves as a stepping-stone for object detection as it divides images in the different shapes and forms the algorithm can find. Object detection does the same for the final step as it recognizes similar objects in the frame. Finally, the classification part takes all the information generated from the previous two and classifies each object detected in the frame. This last part can be extremely precise, as seen in the last image, it can even classify a handbag and differentiate a car from a truck.

10 https://www.analyticsinsight.net/manual-to-object-detection-with-machine-learning/

11 https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

Face recognition is another one of those methods, very important in security, and therefore for our project. Many important companies offer solutions for facial recognition such as Apple with their FaceID as well as Amazon or Google, that offer solutions as cloud vendors. Nowadays, developers in this fields are investigating whether it will be possible to do personal identification based on the geometric shapes of our body, which will increase considerably the accuracy for personal recognition.

The third method is to infer actions from images or videos. This is directly applicable to the final objective of this project to monitor the interpersonal distance. With all the massification of the cities and all the major events happening day in day out, computer vision helps us to digitize all those events to use the data to track behavior, this is not only helpful to find an action in a specific moment but also to analyze crowd flow to develop crowd control which, ultimately, is what we want to monitor and manage.

The last method currently being widely used in security application is modifying and recreating images. This is done using GAN (Generative Adversarial Networks), a specific algorithm within the Neural Network approach, one of the ML methods that perform best. GANs are used a lot for Image Restoration but also have other applications such as generations of images and 3D objects, image-to-image translation and text-to-image translation. This approach reconstructs damaged images. This damage can come from several and different sources such as blurring, obscure images or possible reflections. This approach could be very helpful if we have to deal with a camera feed that is not providing an image as clear as we expect.

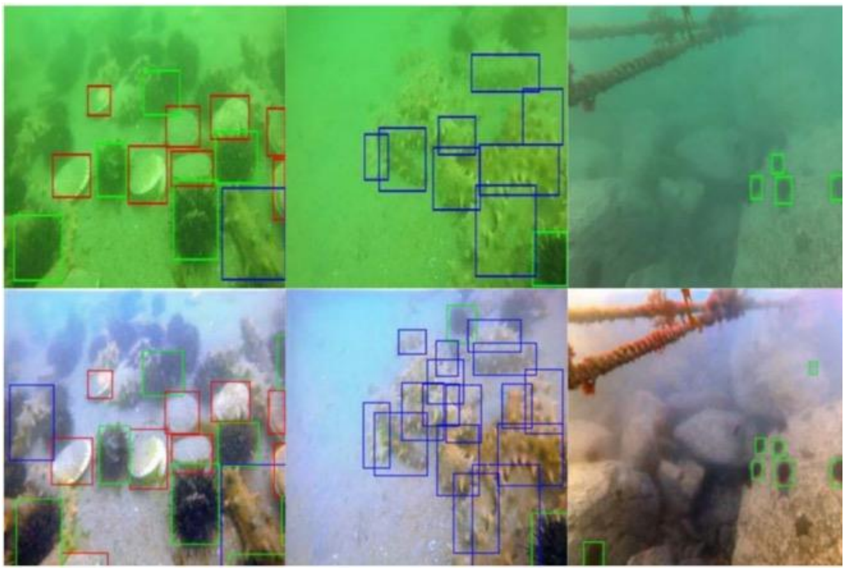


Figure 6. GAN Network Example.[12]

12 https://neurohive.io/en/news/uwgan-underwater-gan-network-for-color-restoration/

After mentioning neural networks, there´s need to explain what these algorithms bring to the table and how they work, at least in a high-level fashion, for now. Neural networks are the base for all the autonomous vehicle algorithms that based their approach on object detection so, for us, it is closely related to our project. Even though autonomous vehicles are far from integration into society, several prototypes have proved to be successful in a controlled environment, but why are neural networks so useful for these environments and, therefore, our project? As mentioned before, classifying and detecting objects is crucial and neural network algorithms are able to work with camera data and sensor data better than any other approach, allowing us to make decisions in near or real-time. They can gather enormous amounts of data (locations, times, pixel information etc.) and process it using deep learning models for detection with a very high accuracy thanks to the high number of training data used. They can also work in low light conditions, so for us, being able to track and monitor movements in low light conditions is crucial, since we want to do it all the time, and, in winter specially and at night, there is no guarantee that the image gets a normal amount of light.

3.2 INTRODUCTION TO IMAGE INDERSTANDING

For all these image understanding, cameras use LiDAR and Stereo Vision techniques. LiDAR is a laser scanner that provides 3D reconstruction of the surrounding environment with a depth and elevation estimation. In the next schema you can see how the system works.

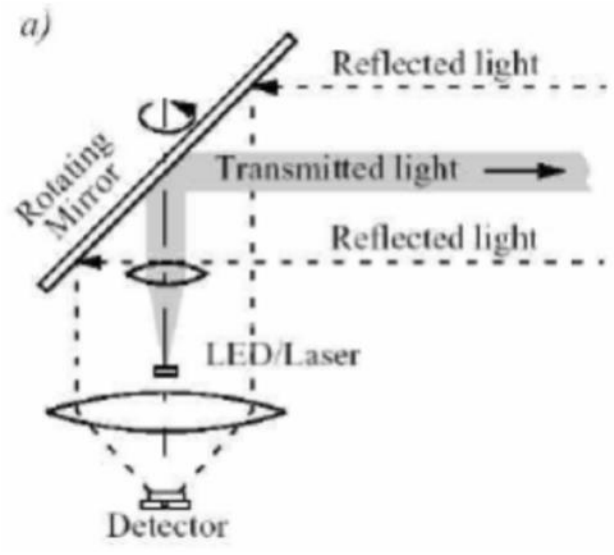


Figure 7. LIDAR mirror detection.[13]

13 http://wavelab.uwaterloo.ca/sharedata/ME597/ME597_Lecture_Slides/ME597-4-Measurement.pdf

The scanner and mirror require an almost perfect precision, so, these systems are very expensive with market prices starting at 1000$ up to 75000$.

A cheaper alternative is Stereo Vision. It uses a multi-camera setup. This allows the system to estimate depth similarly to how humans do it. An image, however, we want to paint it, is a 2D projection of something so, from an external point of view, there is no way we can calculate depth. By triangulating the projections of the different cameras and doing some geometry calculations, cameras can be related to one another to find the distance and, therefore the depth. A simple situation is the one displayed next. That is how OpenCV, one of the most used libraries for computer vision programming, calculates depth, for example.

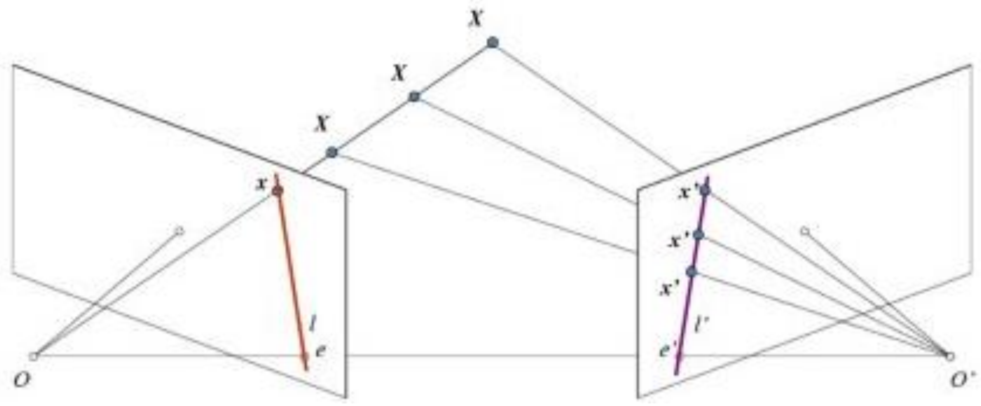


Figure 8. Stereo Vision Processing with OpenCV, [14]

Stereo vision is inexpensive but, until recently, by itself, it cannot work in real time due to the high processing time it takes to resolve the depth geometrically. Now, there are plenty of cameras that can do Stereo Vision with hardware. To reduce the processing time, machine learning and neural networks start to become very useful for image and video processing. Convolutional Neural Networks are used to speed up the processing by sampling an image into a vector and passing it through the network, which assigns probabilities to each value and outputs the probable match.

14 https://docs.opencv.org/master/da/de9/tutorial_py_epipolar_geometry.html

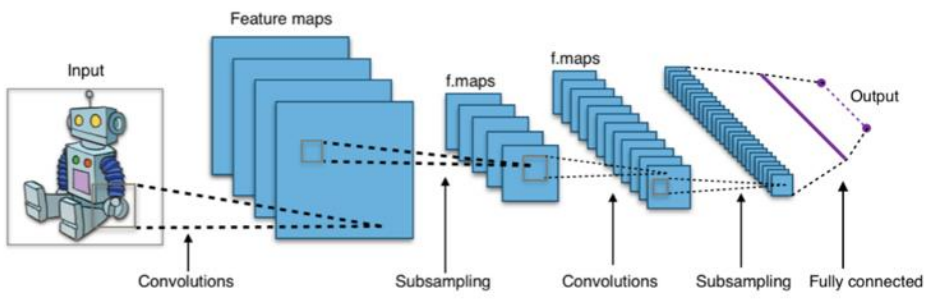


Figure 9. **Convolutional Network Architecture.**

15

With stereo vision, the system creates a projection for each camera and analyze it separately, these are fed into a Siamese CNN that creates at the same time a set of possible probability distributions, then, the output layer provides the most probable distribution for the captured image.

As mentioned earlier, all these techniques, methods and algorithms will be explained with the needed detail later in this thesis, illustrating them with use examples when possible.

3.3 TRADITIONAL TECHNIQUES

To start the immersion into the numerous techniques, approaches, methods and achievements of computer vision, it is very useful to start by knowing at least where current computer vision techniques are based on. With that in mind, traditional computer vision techniques will be discussed next. To start with this, a description of security applications early in the computer vision life can be very useful to later extrapolate to our case. In the early 2000s, surveillance, and therefore security, using computer vision was limited to Video Motion Detection (VMD) and Intelligent Scene Monitoring (ISM) [6]. These techniques were based on pre-processing filtering with Fourier filtering and edge detection. These simple image processing methods weren´t new, but that alone is not computer vision. Computer vision extends image processing to scene content, tracking and object classification. Early computer vision applications with this approach were Number Plate Recognition, Vehicle Tracking and Crowd Analysis. The first technique that was tried as computer vision was called frame differencing. It consisted of differentiating all the scene areas that, with a video paused (A frame), are different from one frame to another. Even though this might be seen as computer vision, it is not, and these are the main reasons: There is no image understanding, there is no concept of target, just different areas and finally, it was susceptible to camera movement and weather conditions. An evolution of this technique was early implemented in PIDS applications and was based in three tasks. The first one was

15 https://commons.wikimedia.org/wiki/Category:Convolutional_neural_networks#/media/File:Typical_cnn.png

object detection, that used the previous image processing technique. The difference between this technique and the previous is that, after object detection, this last one uses deterministic approaches to recognize the objects detected and, finally, after recognizing the target object, it tracks it. This was the first computer vision application; we are in the 1990s. Even though this method proved successful, but there was still an issue, it had difficulties with reduced domain knowledge, that caused that classification problems couldn´t be solved still. The next step in order to solve the problem was the use of neural networks. But what is a neural network?

Early models of sensory processing by the human brain are the inspiration behind neural networks [6]. By coding an algorithm that mimics processes of real human neurons, our newly created, artificial, neural network can 'learn'. A neuron receives input from external sources, or even from other units within the network. It weighs each input and, if the total output is above a threshold, the output is one. So, in relation to our previous problem, a neural network can solve a classification problem, for now, if the target is binary. Incorporating this into the video surveillance, neural networks perform low level processing of pixels and then perform classification on the outputs of these preprocessed pixels. This approach is highly adaptive, which makes it perfect for outdoor cameras processing. How can a neural network learn? A network learns by changing the weights slightly every time we do the processing, which we do on numerous occasions, it is called training. The weights are tuned every time the classification is improved.

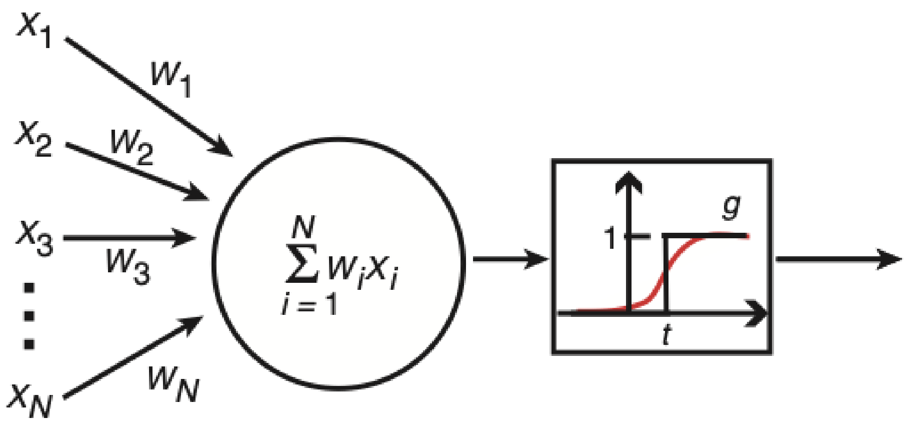


Figure 10. Neuron classifier[16]

The problems that are not linearly separable can be also classified with what is called a hidden layer. This, at a high level, is another threshold unit that its mission is to do partial classification. After that, the output level leverages each partial classification and comes up with a final prediction. These are called feed forward neural networks and are the first type

16 http://people.binf.ku.dk/~krogh/publications/pdf/Krogh08.pdf

of neural networks that appeared. The activation function can be changed into other function that can classify the objects more accordingly to what we want.

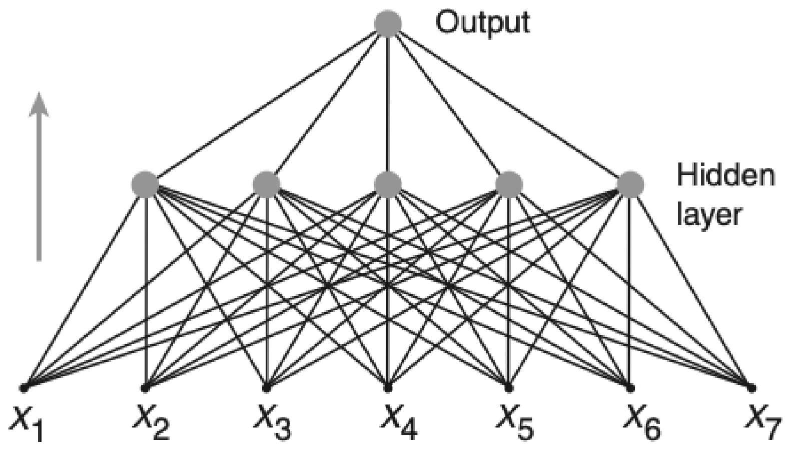


Figure 11. Feed Forward Neural Network[17]

This sounds great for the surveillance video issue, but testing proved that the neural classifier was still one order of magnitude worse than other forms of sensor technology. For now, in the 1990s current technology is not able to process all the contents a video gives.

Entering the 2000s, two approaches were used to do a more robust and flexible tracking. These two techniques are Kalman filters and P2DHMM (Pseudo-2D Hidden Markov Model), that actually is used to feed the Kalman filter [7]. These two algorithms work together, making it possible to track people even if the background has moving objects. The main advantages of using a statistical model approach such as P2DHMM is that, a priori, it is able to recognize some human body shapes and that it can learn the features only relevant to the problem. For now, the system does not rely on motion information, so this works for tracking people even if they are not moving.

Kalman filters provide estimates of a variable given the measurement observed over time. It has two stages, prediction and update. The prediction stage estimates a new measurement from the output of the update stage. The filter is recursive, so a stopping constraint needs to be specified. These filters are widely used to this day to track moving objects.

As mentioned, the Kalman filter provides all the information that the P2DHMM uses and needs, but the input of the Kalman filter also takes the output of the P2DHMM. This allows the shape detection procedure to improve in an iterative mode. Therefore, the process relies only on shape and, optionally, color. With this approach, the system can track a person in front of moving cars because the system only focuses on the person's shape and therefore, motion.

17 http://people.binf.ku.dk/~krogh/publications/pdf/Krogh08.pdf

How the system works within itself is as follows. P2DHMM generates a measurement vector. This will be the input of the Kalman filter, and its components are the center of gravity of the person detected. At first, the system processes the frame by frame and, focusing on detecting the edges, partitions the image in 2D blocks. The parameters of the system consist of the transition and output probabilities of the various HMM states.

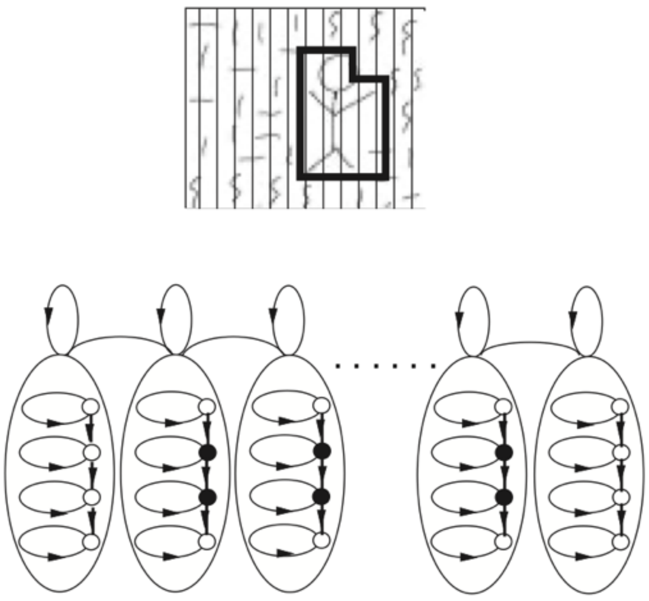


Figure 12. Two-Dimensional Object Detection with P2DHMMM[18]

Combining the P2DHMM output with the Kalman filter and the feedback into the next P2DHMM, it improves the estimation of the detection. It works the following way. The box where the human shape is estimated to be, which is determined by the Kalman filter, is enlarged by a factor of 1.5. This box is then used as the input in the P2DHMM, that focuses its Viterbi search within this box. This allows the segmentation procedure to be much more precise in the next estimation. In each state of the P2DHMM, weights can be introduced in order to influence the importance of diverse features. This is the first approach that has proven successful to track people in arbitrary environments, and it is a good approximation of what neural networks and AI will do in the future by training, back propagating errors and doing convolutional structures.

In a paper developed by the Microsoft Research Vision Technology Group in the early 2000s [8], they established the characteristics and rules that a real-life vision-based tracking system must comply with and respect, and to me, it still applies for our case now.

In this paper the project was designed to track people inside a room. They used stereo cameras, as they make segmentation easier, as explained before. To segment the human figures, they performed background subtraction in depth and color. One of the benefits of

18 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=780267

using the stereo cameras was that they can cope with changes in illumination and shadows better than other types of cameras. After analyzing the depth data, they detect humans by grouping the pixels with similar disparities, forming blobs that are grouped into people shaped clusters. Random outliers and blobs are eliminated by evaluating the first two eigenvalues of its covariance matrix. The surviving clusters are likely to represent people. Tracking is made by updating the coordinates of the validated clusters.

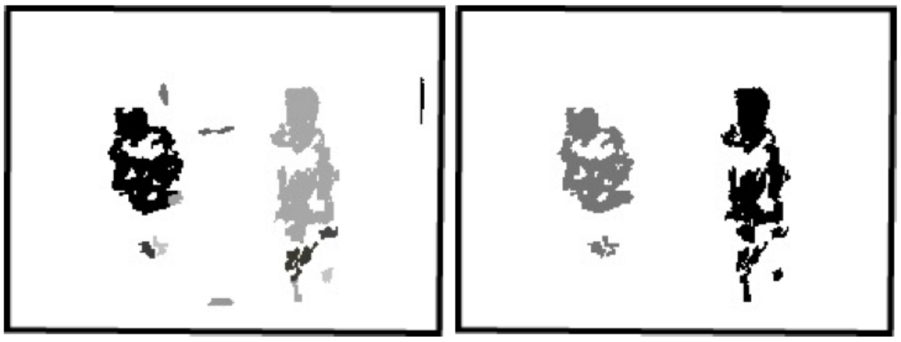


Figure 13. Human Shaped Blobs Generation[19]

This system proved to correctly detect and track 3 persons in the room.

In the matter of tracking and detection, this is not a very good approximation to what this project is about because they do not use AI and neural networks as we intend to do since it was still in the early 2000s but, in terms of the characteristics and basics of what a vision-based system should do, it can be very valuable, and it is a good basis for what is going to come in the future.

The rules portrayed in the paper for a vision-based tracking system are the following, as well as each of the relationship with our project [8]:

1. Maintain the location and identity of people. In general, behaviors of the room are more compelling when this data is known accurately. It is important that throughout the time that each person is detected in the frame that the application does not lose it in order to do a correct tracking of each individual and not mix them up. Occlusion may happen between two persons and mixing up their identifications and their tracking can´t happen.
2. Run at reasonable speeds. Not only do the system needs to run at a reasonable speed but also do it in near real-time, taking into consideration the relativity of what real-time is, in order to be able to make decisions instantly and avoid further problems.
3. Work with multiple people. Of course, in a public environment there is no point in tracking one individual at a time.

19 https://ieeexplore.ieee.org/abstract/document/856852

4. Allow creation and deletion of people's representations. It is generally impossible to predict who will be entering a room. For our system this will relate to flexibility. Every time a new individual enters the scene, a new personal instance needs to be created for him, especially if personal identification is done.
5. Work with multiple cameras. No camera can see around corners, so multiple cameras are required for tracking people in general rooms. Not only in rooms, in outdoor environments we will be using multiple cameras and perspectives for the tracking and detection.
6. Use cameras in the room. Using cameras looking down from high overhead simplifies the tracking problem but is impractical in most rooms. Also, in outdoor environments, having an overhead camera is nearly impossible, so we need our system to detect and track individuals from more complicated perspectives in order to make our system as scalable as possible.
7. Work for extended periods. Not only extended periods, always.
8. Tolerate partial occlusions and variable postures. This is crucial because, especially in outdoors environments, objects can come across and in front of individuals being tracked, but we cannot lose the recognition and tracking of said individuals because in that case, monitoring is not reliable.

Now that an introduction to the state of computer vision in the early 2000s has been presented, we jump forward 5 years and see which techniques were used between 2005 and 2010 to do crowd analysis. Early in the decade, the main problem was where we left off with the Microsoft Project, which is with occlusions and different postures. The applications where this crowd analysis is used are several such as crowd management in different mass events, for safety prevention, public space design, visual surveillance, for automated detection of anomalies and in intelligent environments.
For all these applications, conventional computer vision techniques were almost obsolete, at least for these cases of crowded analysis which will be where we may be working most of the time as social distance in non-crowded spaces is way easier to maintain. Crowd dynamics is a very important subject for our further analysis. Crowds are classified taking into consideration 3 different characteristics:

- Image space domain: In this domain, a crowd is identified when the density of people is above a certain threshold, normally selected when density is sufficiently high to not allow individual identification.
- Sociological domain: In this domain, we take into account the behavior of the crowd.
- Computer graphics domain: Crowd simulation techniques to mimic a real crowd behavior.

All these models and studies have been able to depict some characteristics of crowd behavior that could be very useful, in particular for monitoring the distance [9].

- *Least-Effort Hypothesis*: People try to choose the least-effort route to reach their goals. For our case, this can be an issue because when individuals try to reduce time and space in their routes, the minimum distance between individuals is probably going to be breached.

- *Lane Formation*: It takes less effort for people to follow immediately behind someone who is already moving in their direction than it does to push their own way through a crowd. This can be a blessing in disguise for this project. Theoretically if people form a line in their routes, tracking could be easier for us but, with some perspectives, measuring the distance between people can be much more complicated if they form a line. Also, when in line, occlusion is very probable, difficult the detection of masks and the possible identification of individuals.
- *Bottleneck Effect:* This behavior describes a very obvious effect, in which people change velocity as a function of both density of people and restriction in the environment. If speed changes, it is obvious that, in most cases, minimum distance will not be preserved.

Combining the previously proven successful Kalman filters with geometrical correction the results improved. This is because of the changes in the dimensionality of a person depending on the camera perspective. This approach was only focused on indoor environments with up to around 30 people. To improve the result and expand onto the outdoors environments a new method combining a neural network approach with statistical approaches like Bayesian networks were able to detect high density situations and provide new analysis about the flow of the crowd. For classifying, multiple approaches use Bayesian classifiers. These classifiers were used, at first as tools for crowd counting and made use of Poisson regression models as well. Poisson regression models the noisy output of a function as a random variable, and, adopting a hierarchical model, which extends the Poisson regression to a Bayesian setting. Analyzing the results of the combination of both methods, the results published have proved that these methods achieved success not only for counting people but also detecting the direction of their walk [11].

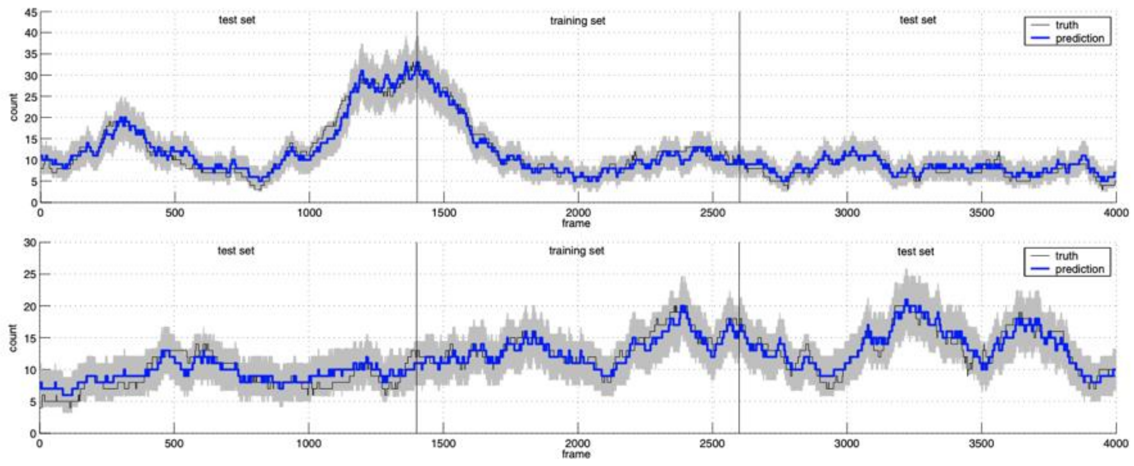


Figure 14. Crowd Count tally over the Time[20]

20 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5459191

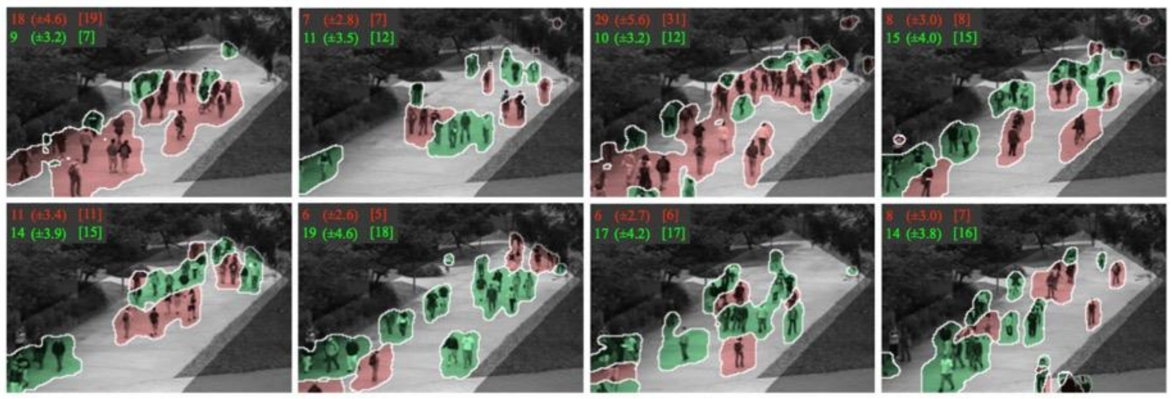


Figure 15. Crowd Counting Examples on Each Frame[21]

Texture level analysis has been used to estimate the number of people in scenes, using, texture analysis methods like Grey Level Dependance Matrix, Straight Line Segments and Fourier Spectrum and then classification methods such as Neural Networks, using basically Self Organizing Maps, Statistics, fitting polynomial functions and SVMs to relate textural features with the density of the scene. For the texture analysis methods, gray dependence matrix is a statistical method that is based on the probability of a gray level in two different pixels. Straight Line Segments is a structural method that consists in using first-order statistics of straight-line segments extracted from the result of a Hough transform. Finally, the Fourier method is a well-known method based on the spectrum of the textures. In this approach, the smoothness of a texture is linearly dependent on the spatial frequency of that texture.
For classification, neural networks have already been used extensively, but one of its different approaches hasn't been mentioned yet. Self-Organizing Maps are used in multiple applications for classification purposes and crowd monitoring is not strange to it [13]. This model implements non-linear projections of high-dimensional spaces onto low-dimensional maps. The n-dimensional vectors hold values of the synaptic strength of the network. For classification, statistical methods such as the Bayesian classifier mentioned before are also used for crowd monitoring, not only for counting. Before we expose the combination with Poisson distributions, Bayesian classifiers are also used with Gaussian distribution, using the density probability function of one class against the probability of said class. Fitting functions used for classifications are no more than polynomial functions of different order whose coefficients are estimators determined by previously training the model and then comparing the error of each combination of coefficients until the result is optimal.

Support Vector Machines have been used also several times as classifiers for crowd monitoring tasks after doing training in the corresponding model [14]. To be able to use the classifier we need to feed it something to classify. So, first of all, an initial calibration of the scene is needed. This is used to build a density map of shapes within the image, a common method is to split the image in rectangles and calculate the density in each rectangle. After

21 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5459191

the image is divided, we carry out the density estimation of each rectangle and the design of the density map of the complete image. The objective of this is to make use of previous processes mentioned in this document to feed the result into the SVM. These processes are, in order, image preprocessing, feature extraction and then the training for the SVMs. For feature extraction techniques such as the previously mentioned Gray Level Dependance Matrix or Straight-Line Segmentation are used, these techniques provide the predictors needed for the training phase of the Support Vector Machine. Finally, we can train the classifier. To do this we use as inputs the density estimates calculated previously and the texture features extracted and use them to predict an output. SVMs have two main advantages for this, which are that it uses very few tunable parameters and minimizes structural risk. For the training, the same as any ML method, a database of positives and negatives needs to be used and then fed into the SVM to construct the classifier.

Result shows that for the Gray Level Dependance Matrix, the optimal result taking into account the parameter correctness is the statistical approach, the Bayesian Classifier, as well as when using the straight-line while, when using a Fourier Spectrum analysis, Neural Networks prove more accurate.

This proves that, with the passing of the years and the continuous development in AI and Machine Learning, neural networks are rapidly becoming the best method for predicting and classifying in crowd monitoring tasks, from one order of magnitude difference 7-8 years ago to being better performer in some situations in the 2010s.

3.4 MODERN TECHNIQUES

Now that the state of the art on traditional techniques has been covered, more modern techniques are going to be explained next to provide a complete scope of where computer vision sits now.

Multiple high-tech companies like Google, Tesla and Amazon have spent a lot of money and resources recently researching computer vision and the current techniques, while having a similar basis as the traditional techniques, the complexity and flexibility of the new ones is substantially higher.

3.4.1 SEMANTIC SEGMENTATION

To start with the modern techniques, we start with object detection, the first task to carry out in a computer vision project after feature extraction, but that is carried out by CNNs, that have this process incorporated, but the basis of object detection often relies on semantic segmentation. There are numerous methods to capture images for computer vision like SONAR, the above-mentioned LIDAR and stereo vision and now, monocular cameras. Monocular cameras are widely used now because they can be separated in appearance-based and motion-based cameras [15]. Using both types together make object detection much more accurate. A very common method is semantic segmentation, and a technique developed by University of Cambridge researchers that has proven successful is de SegNet Encoder-Decoder. SegNet is a deep encoder-decoder consists of four layers of encoders and their corresponding layer of decoders, followed by a pixel classifier.

The encoder uses convolutional layers with ReLu non-linearity and is followed by max pooling and down sampling, as seen in the image below. The decoder up samples the output of the encoder to obtain the input image. The Gradient Descent method is used to train the network [16], [17].

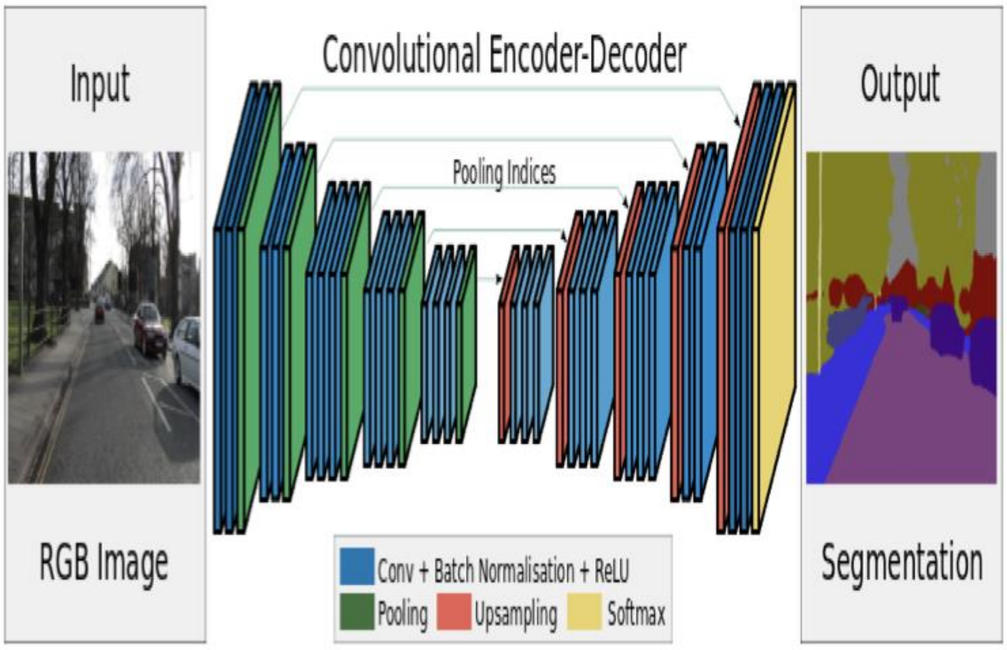


Figure 16. SegNet Architecture[22]

In a very few words, you have read a lot of new concepts that need to be introduced. First of all, the convolutional neural networks (CNN) like SegNet uses. CNN have been mentioned before but it is necessary to explain them further as they are key to our project.

CNNs are a Deep Learning algorithm [18] [19]. Deep Learning is a subfield of Machine Learning that is based on the functionality of the human brain and its objective is to try to emulate it. Therefore, it is based on artificial neural networks, where CNN is just one of the algorithms in Deep Learning. It is separated from common machine learning algorithms because its performance with large amounts of data is significantly better than other learning algorithms such as Random Forests, SVMs or Gradient Boosting.

22 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8126154

Convolutional Neural Networks, for our case, take an input image, assign importance to the different parameters and differentiate one from another. While in traditional techniques the preprocessing is more archaic and tedious, the preprocessing required for CNNs is much lower. The difference with a common Neural Network is that, in NN, we can just take a matrix of pixel values and turn it into a 1D vector to use as input. In CNN we cannot do that because it can also capture the spatial and temporal characteristics of an image with the appropriate filters. The convolutional part is that the objective of the CNN is to reduce the images into a form that is easier to process without losing feature information in order to provide a good prediction. CNNs take the values of each matrix element (Normally 3, each from the corresponding RGB channel) in groups and use convolution to put it in one cell of the output matrix, the goal is to extract high-level features such as edges, as mentioned earlier.

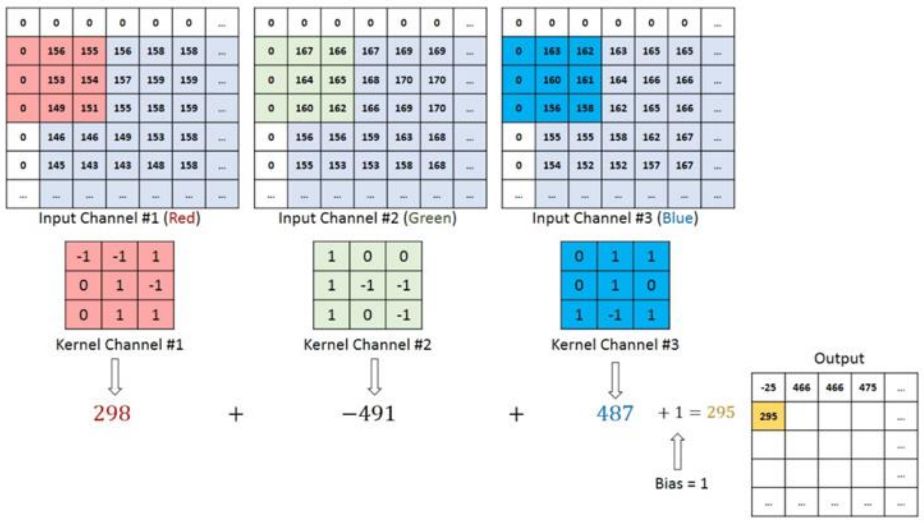


Figure 17. Convolution Process on RGB Channels[23]

Combining multiple layers, we can establish the function of each level in order to be able to obtain more feature information. In the first layers, normally, low-level features like color and edges are detected and in the next layers, the architecture adapts to capture the high-level features. After the convolutional layers, there is a pooling layer with the objective of reducing the computation power required to process the data from the output of the convolutional layer.
The output of the convolutional part of the CNN serves as the input of the neural network, whose inputs are the features extracted from the image in the convolutional part. This neural network is a fully connected layer and allows the algorithm to learn non-linear combinations of the features. Now, we can flatten the matrix into a 1-D vector.

23 https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

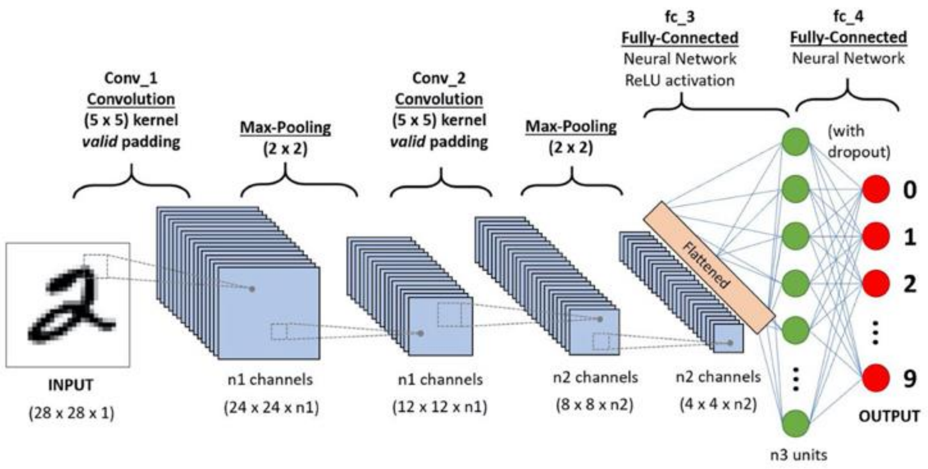


Figure 18. Convolutional Network Architecture[24]

Another technique for semantic segmentation is using a conditional random field (CRF) [15]. These models match potentials between pairs of pixels in an image. Since the computational cost and memory requirements to do this is very large, this method has only proven successful in small images. Combining this technique and CNNs is another method of achieving semantic segmentation. The CRFs allow the model to capture fine details of the RGB pixels.

Another technique using CNNs is Deeper CNNs. This is using several CNNs in parallel. This architecture called ResNet is based on three stacked layers. The first layer is a common CNN, the next layer only processes the residuals from the previous stream in order to be able to process high-definition details. The Dense convolutional network processes everything, the original input and the output of the previous streams.

Some of this different network can be used as individual layers in a deeper model and combine it with the others.

3.4.2 OBJECT DETECTION

The next step in modern techniques is object detection. Finding potential objects is done by finding the overlapping pixel regions. As exposed before, detecting pedestrians reliably is complicated due to the highly variable motion and posture. This is often done with video cameras, using the visible spectrum during daytime and the infrared spectrum during night-time. Thermal cameras are used also because they provide a temperature measure, which allows differentiation between warm objects such as pedestrians and cold objects such as

24 https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

concrete or vegetation. Classical techniques like the preprocessing of images along with SVMs have been explained before as traditional techniques. Nowadays all the state-of-the-art detection systems use CNNs to learn expressive features from large datasets. The Part-Based approach is a semi-traditional technique that combines the idea of splitting the complex appearance of human figures to train the models. The Deformable Part Model tries to break down the complex appearance of objects into easier parts. Then, an SVM is trained and use as the classifier.

Using Deep Learning, other methods and techniques have been developed. The main method is the above mentioned CNNs, but some applications use a variation of them. One of the most important ones is the Region-Based CNNs. They solve the problem of selecting a huge number of regions in common CNNs by generating many region proposals by selective search and generating a feature vector classified later by an SVM. Selective search uses a greedy algorithm to recursively combine similar regions into larger ones and use them as final candidate regions. These are very computationally expensive so a variation of the Fast R-CNNs was proposed, and it improved the results by using a single-stage training algorithm that classifies and refine locations at the same time [15].

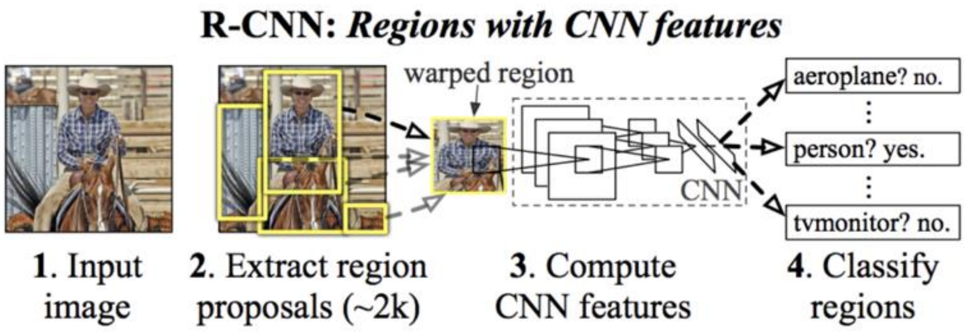


Figure 19. R-CNN Architecture[25]

Fast R-CNN feeds the input image to the CNN to generate an initial feature map. From that map, the algorithm identifies the region proposals and warps them into squares using the pooling layer. It is faster because the convolution operation is done only once. A new variation called Faster R-CNN that does not use a selective search algorithm but lets the network predict the region proposals reshaped later by the pooling layer. This last variation allows us to do real time object detection [15][20].

25 https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

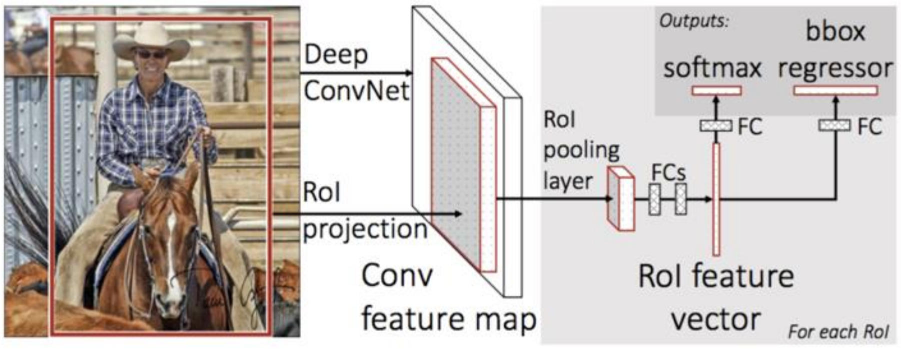


Figure 20. Fast R-CNN Architecture[26]

Deep Belief Networks, Deep Boltzmann Machines and Stacked Autoencoders are other common algorithms used in computer vision for object detection [15],[21]. Deep Belief Networks (DBN) are probabilistic generative models. What they provide is a probability distribution of both data and labels combined. They are based on Restricted Boltzmann Machines which are undirected graphical models with stochastic variables. Each variable is connected to each hidden variable. DBNs use a greedy learning strategy to initialize the network. What these models do is, graphically, extract a hierarchical representation of the training data. The process, is the following [21]:

1. Train the initial layer.
2. Use the first layer output, which will be a representation of the input as the input of the second layer.
3. Train the second layer as an RBM again.
4. Iterate the two previous steps as long as needed.
5. Fine-tune all the parameters with respect to the criterion we want to choose.

There are two main advantages of the DBN approach. The first one is that it solves the problem of the appropriate selection of parameters. Secondly, there is no requirement for labelled data since it is an unsupervised method. As a disadvantage, DBNs do not take into consideration the two-dimensional structure of an input image.

Deep Boltzmann Machines are another variation of the RBM as a deep model. The difference with DBNs is that DBMs layers are completely undirected, as seen in the next image. This allows the model to more controllable versions of the model thanks to the possibility of doing a more appropriate selection of interactions between hidden and visible units. Other difference with DBNs is that DBMs train all the layers jointly. One of the advantages of DBMs is that they can capture many layers of complex representations. Those representations are appropriate for unsupervised learning since they can be trained on unlabeled data, but it is also possible to fine tune the parameters for a specific task. Other

26 https://arxiv.org/pdf/1704.05519.pdf

advantages that the lack of direction in this model provides is that it has some feedback, incorporating some uncertainty in a more effective way. The main disadvantage of this model is that it has a very high computational cost that makes these models unusable when it comes to large datasets.

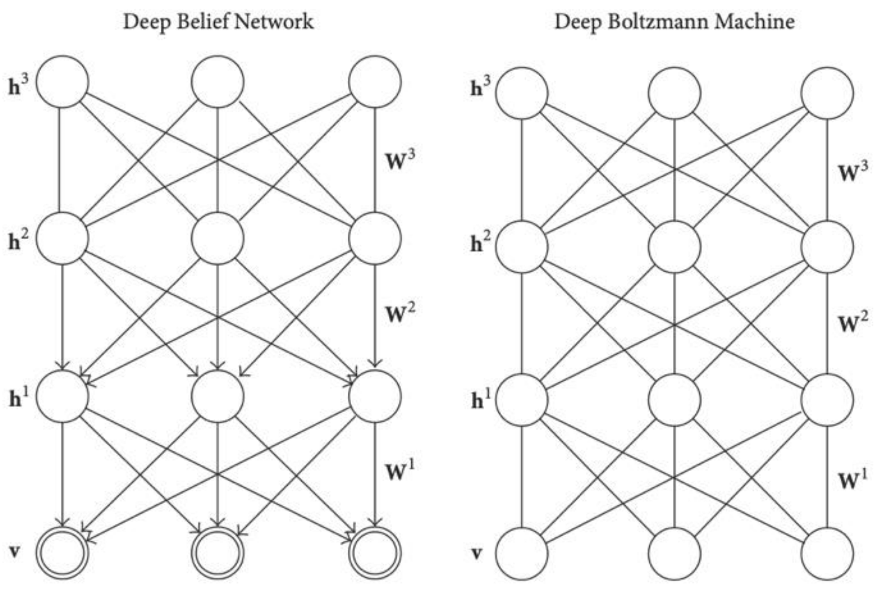


Figure 21. DBN and DBM Architectures[27]

Another technique for object detection is Stacked Autoencoders [21]. The autoencoders used are denoising autoencoders. These autoencoders use the uncorrupted input as the target for the reconstruction, trying to predict the corrupted values from the uncorrupted ones. It is by stacking denoising autoencoders how we implement the Deep Learning approach. The output code is fed into the next layer. Each layer is trained by minimizing the error when reconstructing the input. A second stage of training is done where the parameters are fine-tuned. In this stage, it is a supervised method, and the goal is to optimize prediction error on a supervised task. A third layer is added and trained like a multi-layer perceptron. This is a logistic regression classifier used on the output code of the last layer. One advantage of this approach is that the unsupervised part of the algorithm allows almost any parametrization of the layers.

27 https://www.hindawi.com/journals/cin/2018/7068349/

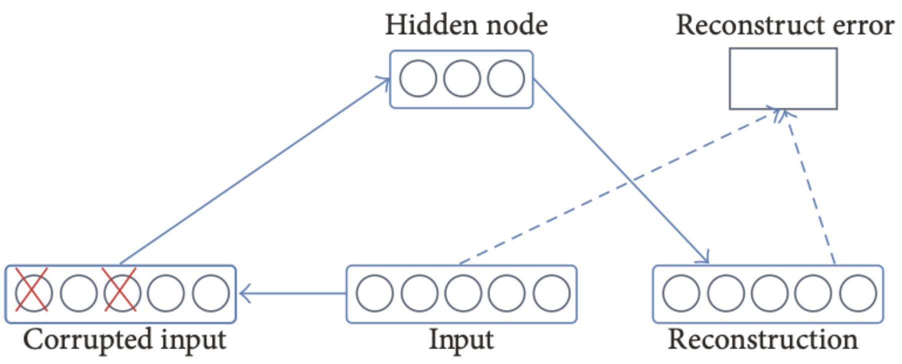


Figure 22. Denoising Autoencoder Structure[28]

YOLO is a model developed for object detection that uses the target detection as a regression problem for a separate target box and its category [21]. A single neural network predicts the confidence of the box and the category. Initial YOLO implementations could only detect 19 objects and had high localization error. An evolution appropriately called YOLOv2 follows the following process, implementing batch normalization on all the convolutional layers and a high-resolution classifier [22]:

1. At first, the image is divided into a grid. If the object we want to track is in a grid, it is responsible for detecting the object. Each grid cell predicts the detection boxes and its confidence.
2. For each detection box, YOLO establishes 5 values, the coordinates of the center, height, width and confidence, all of these related to the box.
3. Prediction of the pedestrian probability for each grid
4. The probability is multiplied by the predictive value of each box, called IOU, which is the overlapping area between the prediction box and the ground truth box. This operation gives the confidence of the class as a result for this box.

There are some variations in the application of YOLO to detection. One of them is the YOLO-R network which is a recursive application of the YOLO algorithm. It combines convolutional layers alongside pooling layers (Using the Max criterion) in the same way as CNN and reorganization layers. This method increases the performance of YOLO by a small margin in terms of precision and recall [22]. The main advantage of YOLO is that it is faster than other methods, including Fast-CNN. The problem is that it can only predict 20 classes. A simplified schema of YOLO is proposed next.

28 https://www.hindawi.com/journals/cin/2018/7068349/

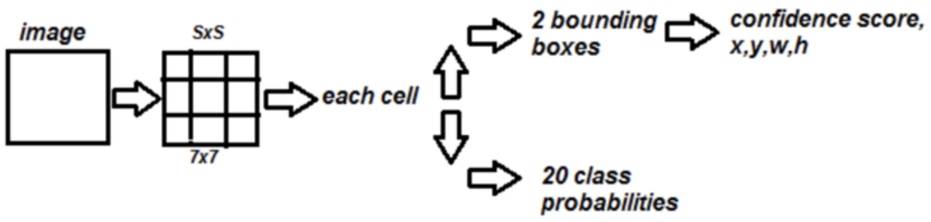


Figure 23. YOLO Schema[29]

YOLOv2 is faster than other methods, but it hasn't proved successful for real-time video object detection for environments with limited computational power. This is the reason why another approach with YOLOv2 called Fast-YOLO was tried by researchers in the University of Waterloo [23]. Fast-YOLO is divided into two stages. The first one is an optimized YOLOv2 architecture and the second one a motion-adaptive inference. The motion-adaptive inference determines if deep inference is needed for a particular video frame, which is not always necessary. As seen in the following schema depicting the structure, if deep inference is not needed, Fast-YOLO uses the stored reference class probability map. Experimental Fast-YOLO results show a more compact architecture and a faster run-time.

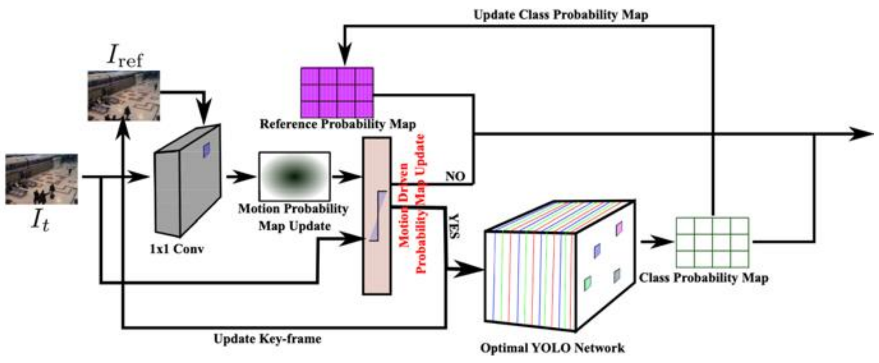

Update Class Probability Map
I_{ref}
Reference Probability Map
I_t
1x1 Conv
Motion Probability Map Update
Motion Driven Probability Map Update
NO
YES
Class Probability Map
Update Key-frame
Optimal YOLO Network

Figure 24. Fast-YOLO architecture[30]

LSTM (Long-Short Term Memory) networks are also very popular in Deep Learning tasks, such as object detection [15][26]. These networks are a modified version of recursive neural networks. The difference is that these networks make it easier to remember past data, an aspect at which recursive CNNs have troubles. The main advantage is that they can solve the problem of the vanishing gradient. LSTMs are able to classify, process and predict given some input data. It trains the model using backpropagation. The structure is as follows:

29 https://medium.com/@amrokamal_47691/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899

30 https://arxiv.org/pdf/1709.05943.pdf

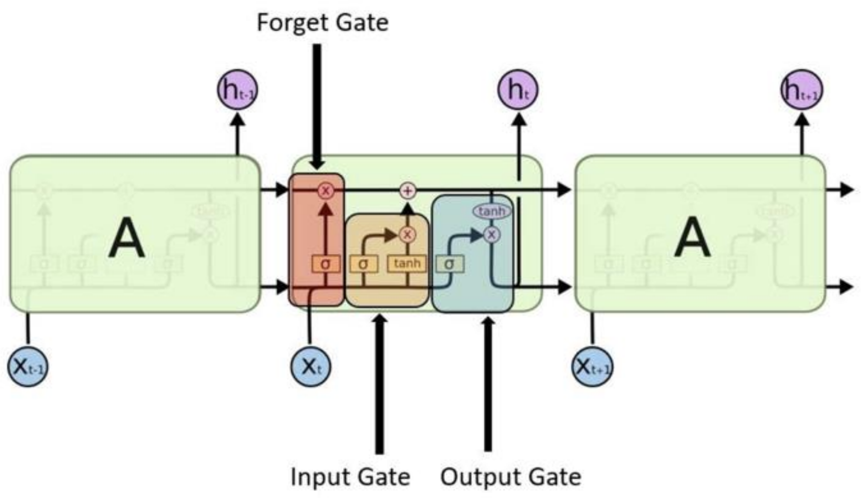


Figure 25. LSTM Cell Architecture[31]

The input gate is used to discover which input value should be used to modify the memory. Using different activation functions like *sigmoid* or *tanh*, weights and importance are given to the values. The forget gate decides which details have to be discarded from the block and does it by looking at the previous state for each number in the cell state. The output gate decides what to output from the memory.

In relation to object detection, LSTMs are often used in combination with CNNs, Fast-CNNs, R-CNNs and with SSD, Single-Shot Detectors. Single Shot Detectors are quite popular object detection methods based on CNNs to detect bounding boxes and provide scores for the presence of objects of that class. The secret of the performance of this method is that contrary to CNNs or YOLO, it performs data augmentation of the original image before feeding it to the network. The combination of LSTMs and SSDs happen as described next. The SSD extracts the objects in the frames and their detection boxes, with the corresponding score. After selecting the objects detected, the vector is fed into the LSTM network. Given the input tensor, LSTM obtains the object regression and association scores, if using a LSTM association network.

As mentioned early in this thesis, Siamese CNNs are one of the more used variations of CNNs [15]. They are used when stereo vision is the input to the model, and can help with multiple perspective problems, lighting problems and image quality problems as well. This basically means that, to simplify it, the network has two images as inputs. Siamese CNNs objective is to calculate the differences between these images and output the result to a single neuron output using one of the many activation functions available.

31 https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e

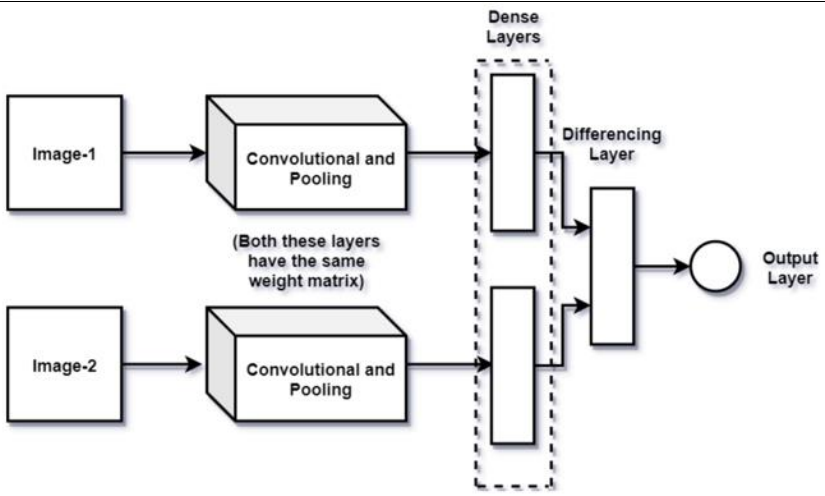

Dense Layers
Image-1
Convolutional and Pooling
Differencing Layer
(Both these layers have the same weight matrix)
Output Layer
Image-2
Convolutional and Pooling

Figure 26. Siamese CNN Structure[32]

The main problem with S-CNNs is that the output extracts fixed representations for each image but not the knowledge of the paired image. This makes the architecture fail in propagating the local patterns necessary to increase the confidence level.

It is obvious why these types of networks are really useful when dealing with different perspectives of an image, differencing between people etc. In object detection, two main approaches using Siamese CNNs have been developed. The first one is a Gated Siamese CNN, used to match pedestrians across multiple camera views. The use of a 'Matching Gate' makes it possible to avoid the problem of the loss of knowledge and the architecture is able to propagate the relevant features and similarities further in the network. The Matching Gate inputs are the outputs of the previous convolutional block. It compares the local features and outputs the mask with the emphasis needed for each feature [15], as it is seen on the next image. [24][25]

32 https://towardsdatascience.com/siamese-networks-line-by-line-explanation-for-beginners-55b8be1d2fc6

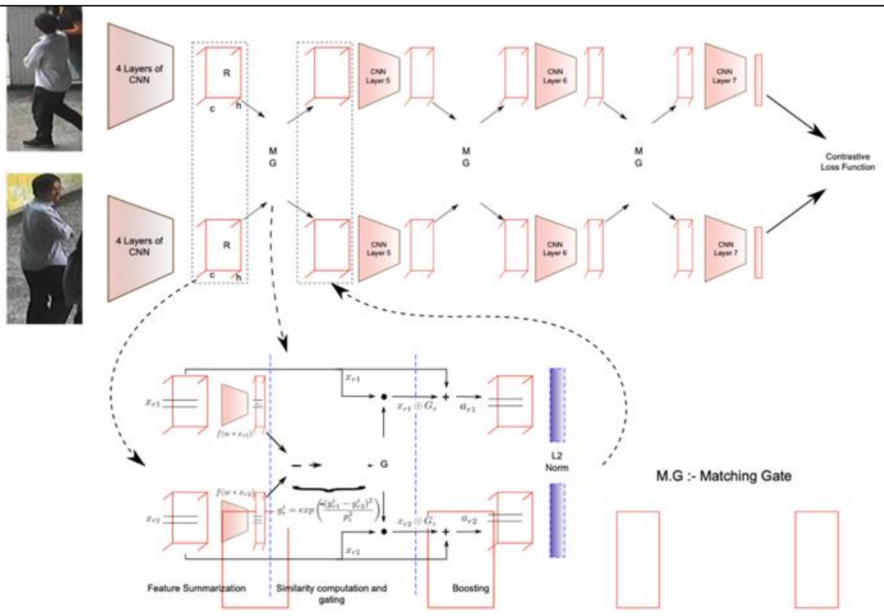


Figure 27. Siamese Matching Gate CNN Architecture [25]

The similarity between features can be computed using one of the many distance metrics commonly used in Machine Learning & Artificial Intelligence such as Manhattan Distance or Euclidean Distance. After the distance is computed, an activation function such as a Gaussian activation function is used to obtain the gate values.

For the same case of multiple perspectives, Siamese CNNs have been also used in combination with LSTMs. As explained before, LSTMs can propagate the features longer in the architecture than regular S-CNNs. Using LSTMs cells enhances the discriminative capabilities of the local features.

As seen throughout this extensive introduction to how the current research in Deep Learning techniques for human detection and predictions is at this point in time, it is, for now, very clear that any variation of a CNN architecture is a very good start for any project dealing with an object detection task.

3.4.3 OBJECT TRACKING

One of the main uses of segmentation techniques mentioned above is the tracking of objects, which happens to be the last stage of a human detection and tracking project. The goal of tracking objects is to estimate the state and position of said objects over a period of time. The difference between this process and object detection is that object detection allows you to process each frame independently while tracking can't do that.
The previously explained Kalman filters are to this day extensively used while dealing with tracking problems. The first technique is called tracking-by-detection. This technique

divides the task into two parts, the first part is detecting the object and the second is associate detections of the same object in different time points. The first method for this technique is using graphs for tracking. Given two images in the same space in different time frames, the nodes of the graph are the detections, and the edges are the spatial and temporal differences between the images. Specifying some constraints and using different programming approaches the network flow can be solved, for example using a k-shortest path algorithm. A second method within this technique is using multiple cues. This approach uses data association to improve the robustness of the tracking system. These cues are used to combine different textures, shapes and depth coming from stereo systems. This tracking-by-detection is the preferred method nowadays in terms of usage and performance.

A second technique is tracking with stereo. This is different from using stereo as the input to the tracking system. This technique estimates at the same time the stereo depth, object detection and the pose of the objects over time using a graphical model. The main method for this technique is Tracking-Before-Detection. Since segmentation is facilitated by depth, in this approach, the segmented classes are directly considered as observation for the tracking. This way, it can track unknown objects regardless of their classification that, in reality, is not necessary for the tracking.

Using Deep Learning for detection, CNNs and LSTMs are the most extensively used techniques. Deep Learning techniques allow us to expand into multi-object tracking. For this stage, RNNs and Siamese CNNs algorithms have been also used in various applications. Hierarchical clustering is commonly used to track similarity in some of these approaches also. So far, the first end-to-end multi-object tracking system was the one exposed in the next image [26].

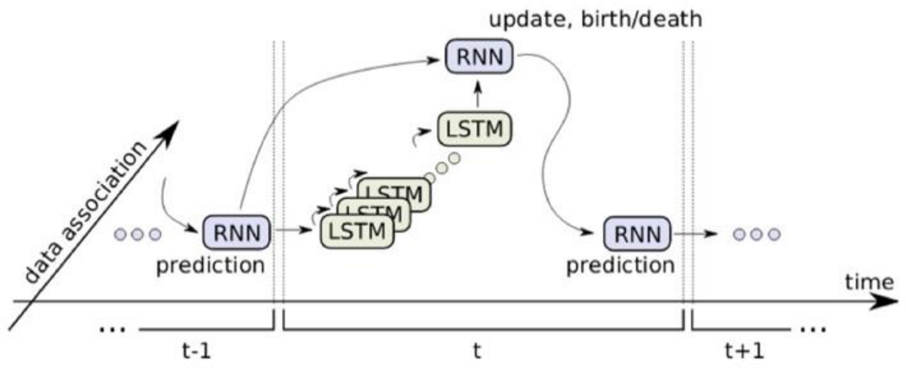


Figure 28. Schema of the Architecture [26]

This model uses RNNs for temporal prediction and update as well as track management. After the RNNs, LSTMs are used to solve the data association problem. The advantage of this division is three-fold. The system can isolate individual components easily and analyze them, the system is modular so replacement of a module does not mean all the system needs to change and finally, it can train each block separately. A more detailed schema is shown next.

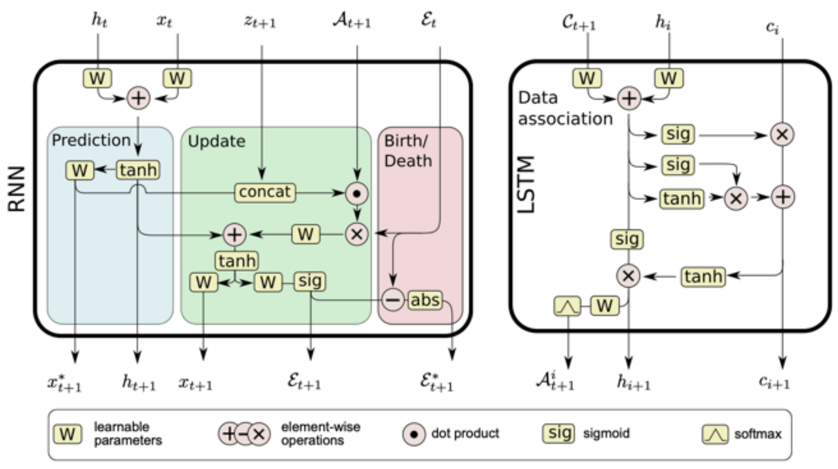


Figure 29. Structure of The RNN and LSTM [26]

The prediction module learns to predict a target motion without measurements. The update part has measurements, so his task is to pair the targets to the corresponding measurements. The Birth/Death part determines the start of the track and the finish based on the parameters passed from the previous parts such as state, measurements and data association. The LSTM uses its temporal functionality to predict the assignment for each target. Using a softmax layer, the LSTM is basically used as the classification module of the system.

Two stage data association has been used also for online multi-person tracking, and therefore, real-time applications. Data association can be used to classify or to detect targets and then feed them into a classifier. A two-stage data association module works as follows. For both cases, the detections need to be fed into them. The first stage solves the assignment between targets and detections. Using affinity values for appearance, shape and position. The position affinity overrides the rest of them, if it is 0, there is no assignment between target and detection, this makes processing speed better. In scenes where occlusions are not long-term and accurate detection measures, it is possible to track people with just the first stage. If there are long periods of occlusion and the person changes motion or position from one occlusion to another, the second stage is needed. The second stage takes the targets with un-confident measures and the detections without an associated target and assigns them to one another. Experimental results have proven the improvement of this method over a one-stage data association model [27].

Another way of using LSTMs for tracking is using different detectors in front. For example, another technique uses a YOLO object detector simultaneously to the tracker, which is composed by a CNN and an LSTM tracker and multiple LSTMs, which are in charge of the data association. LSTMs are so popular for the data association process of tracking because they allow end-to-end fine tuning of all the possible parameters. The tracker can be feeding the data association module at the same time as the detector, not only last like in the previous techniques presented. YOLOv2, which as presented earlier is one of the algorithms that has the best performance for real-time object detection, feeds the results of the detections validated using IoU to the multiple LSTMs in the data association module. The single-object tracker has a CNN and one LSTM unit. Each tracker is trained using Deep Reinforcement Learning, which will be briefly explained later. One of the fully connected layers of the tracker predicts the motion of the video frame and the other computes the confidence. In [28], this technique proves to be very robust and accurate for real-time object tracking.

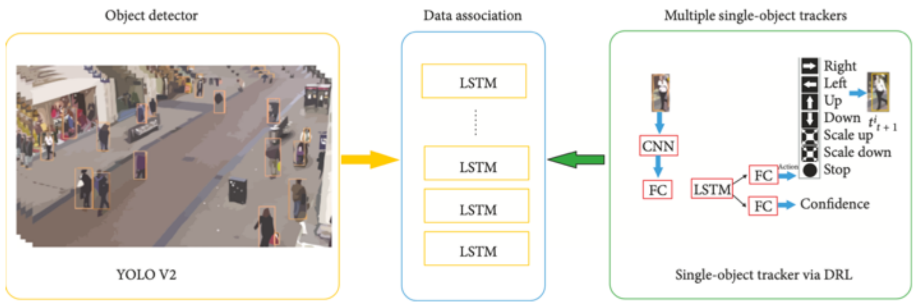


Figure 30. Schema of the Proposed System using YOLO and DRL [28]

Deep Reinforcement Learning is used in AI and Machine Learning frameworks to solve complex sequential decision-making algorithms. The process of a DRL process is presented in the next image. The basic idea behind DRL is that the agent (Single object tracker in this case) is given a reward each time that the action (The association between tracker and detection) proves successful. Two principal methods of DRL have been widely used in RL problems. These two are the Deep Q networks and the policy gradient method. DQNs uses a neural network to do function approximation. This means that given a state-action value produced by the neural network, it tries to learn it by minimizing the temporal-difference errors. Policy gradient approaches is another DRL method that uses the gradient descent algorithm to optimize the policies of the network. These policies are what makes the algorithm decide one way or another. This method has significant advantages to the traditional DRL algorithms. They need fewer parameters to obtain and represent the optimal policy than, for example, DQN architectures.

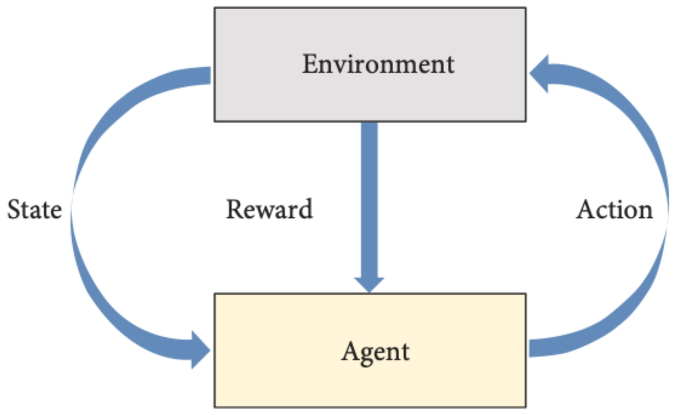


Figure 31. DRL Process [28]

CNNs, as it is seen in the previous techniques are very important for tracking also. Two variations of them that handle all the tracking without any more help are the previously explained Siamese CNNs and the Graph Convolutional Networks.

Taking into consideration the 3 possible structures of Siamese CNNs, the input stacking approach gives a better performance for tasks involving comparison than the other two, that are better for classification (Object detection) purposes as developed before.

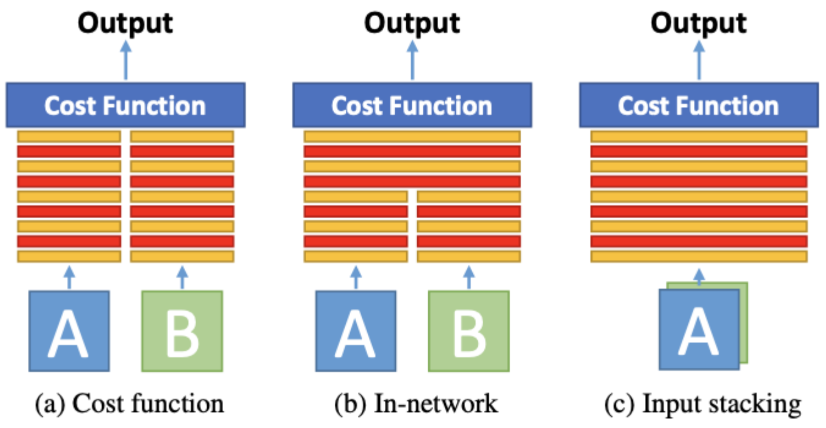


Figure 32. Different Siamese-CNNs Structures[33]

An interesting approach to avoid overfitting with S-CNNs is by generating random false positives, avoiding to overfit a specific type of positives and increasing the generalization capabilities of the system.

Data augmentation is also performed to avoid overfitting. Since the pairs of pedestrian detections tend not to have significant dissimilarities, geometrically distorted images, rotated images etc. introduced as the input increases the variability of the data fed into the architecture. The S-CNN is used in combination with a second stage in charge of extracting the contextual features of the images since the S-CNN is only detecting the pedestrians but not using any temporal and spatial constraints. The architecture is as presented next [24][25].

33 https://www.cv-foundation.org//openaccess/content_cvpr_2016_workshops/w12/papers/Leal-Taixe_Learning_by_Tracking_CVPR_2016_paper.pdf

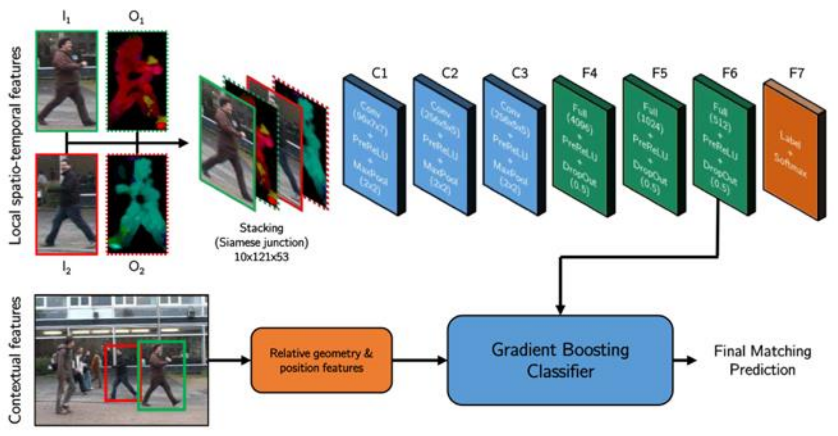

Local spatio-temporal features
I1
O1
I2
O2
Stacking (Siamese junction) 10x121x53
C1
C2
C3
F4
F5
F6
F7
Contextual features
Relative geometry & position features
Gradient Boosting Classifier
Final Matching Prediction

Figure 2: Proposed two-stage learning architecture for pedestrian detection matching.

Figure 33. Model Architecture Proposed[34]

After using S-CNNs for the matching prediction, the tracking can be done using simple linear programming instead of more extensive and computationally heavier techniques such as LSTMs, which still are better performers overall. One of the linear programming techniques to use for real-time tracking is Kalman filters [19]. As seen in the above image, an optical flow version of each image is in the input of the architecture.

Optical flow is the motion of the brightness pattern between two images. It is very useful to use and compute the optical flow difference to train the architectures, for object detection to match the predictions and for tracking. There are several techniques to extract the optical flow of an image. Semantic segmentation is one of them and can be used as the input in deeper architectures as the previous example shows. CNNs are also used for optical flow detection. FlowNet and PWCNet architectures are some of the approaches to it. FlowNet is a convolutional approach that in its first stage contracts the image to extract the important features and then expands it to produce the high-resolution optical flow output. The simplest method of this architecture simply stacks the images. PWCNet uses three layers and, previously, a Siamese network to estimate the optical flow.

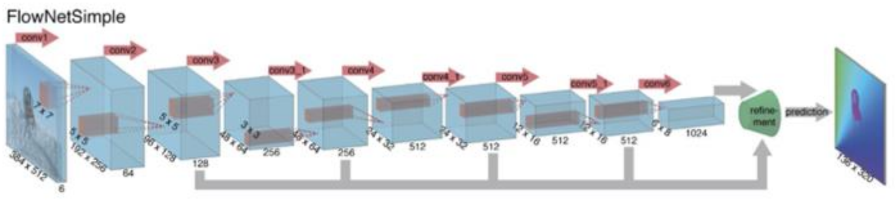


Figure 34. FlowNet [15]

34 https://www.cv-foundation.org//openaccess/content_cvpr_2016_workshops/w12/papers/Leal-Taixe_Learning_by_Tracking_CVPR_2016_paper.pdf

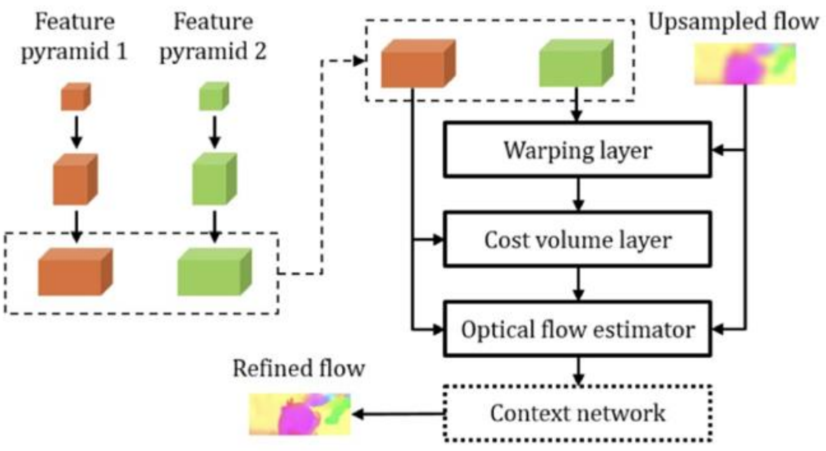


***Figure 35. PWCNet** [15]*

After this brief parenthesis to briefly introduce optical flow, the final variation of CNNs for tracking is the following. Graph CNNs (GCNs) is a segmentation-based tracking method [32]. This is a new approach, different from the tracking-by-detection that has been widely covered. Tracking-by-segmentation segment the object from the background pixel-wise and track the target according to that segmentation result. The advantage of these convolutional networks is that they can extend the architecture to graph-structural data. GCNs are used to measure relation or similarity, therefore, for training, they use both node features and the structure of the graph. The idea is to average all neighbors features to pass them through the network.

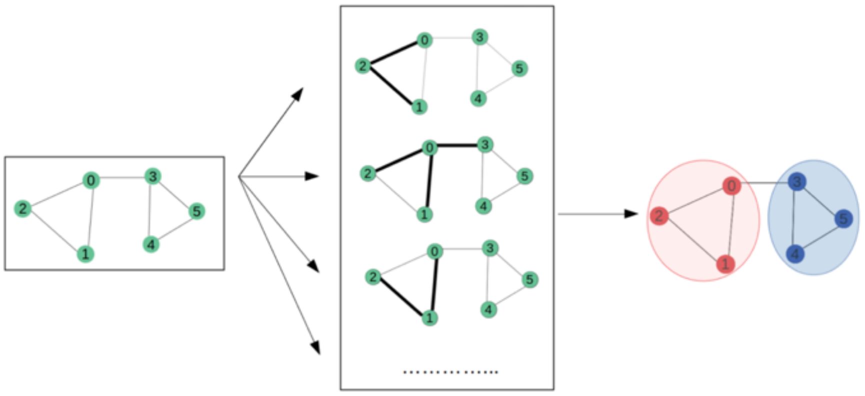


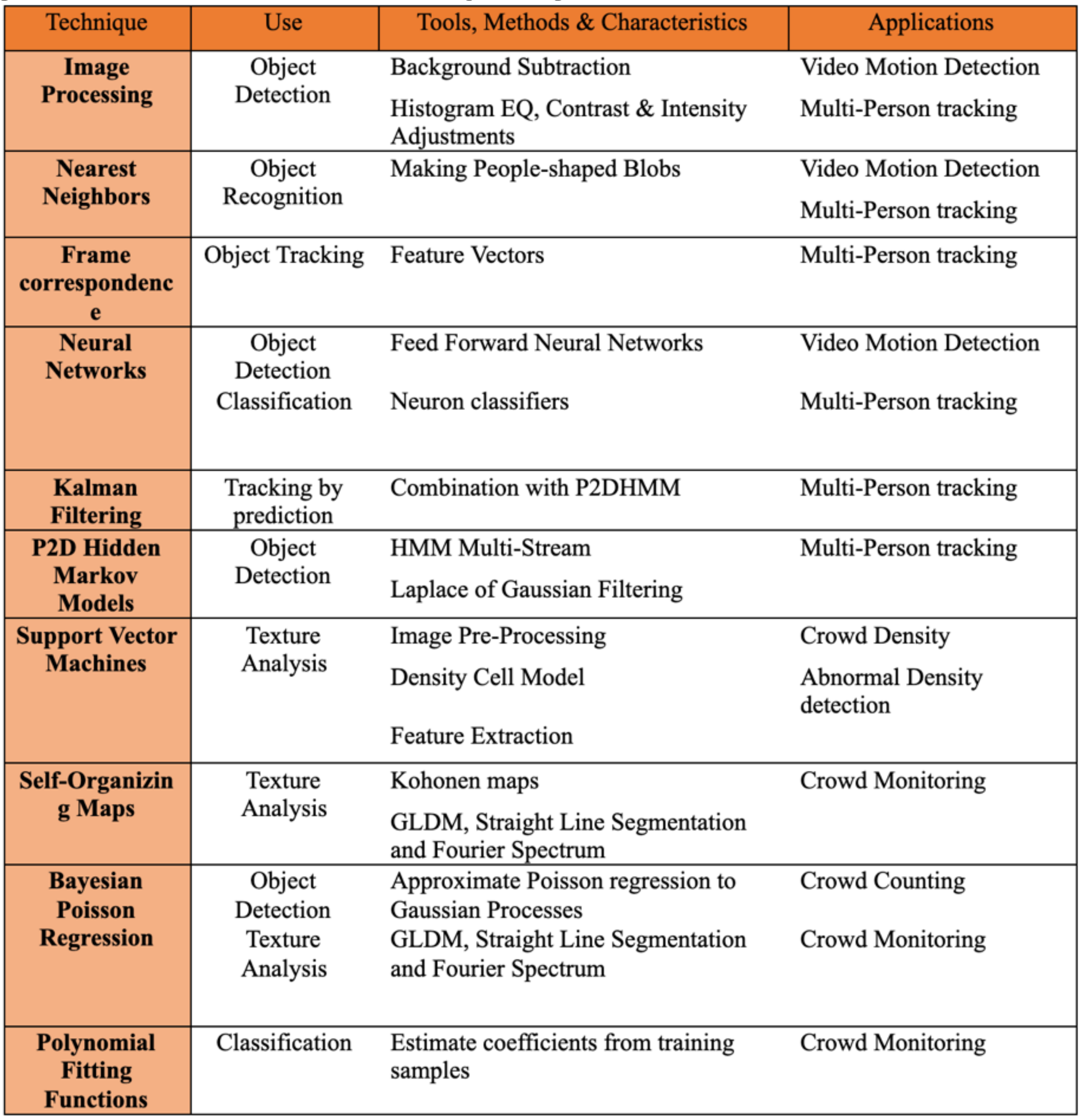
***Figure 36. Graph CNN Process**[35]*

A summary of all these techniques above mentioned and explained is presented in the following pages.

35 https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b

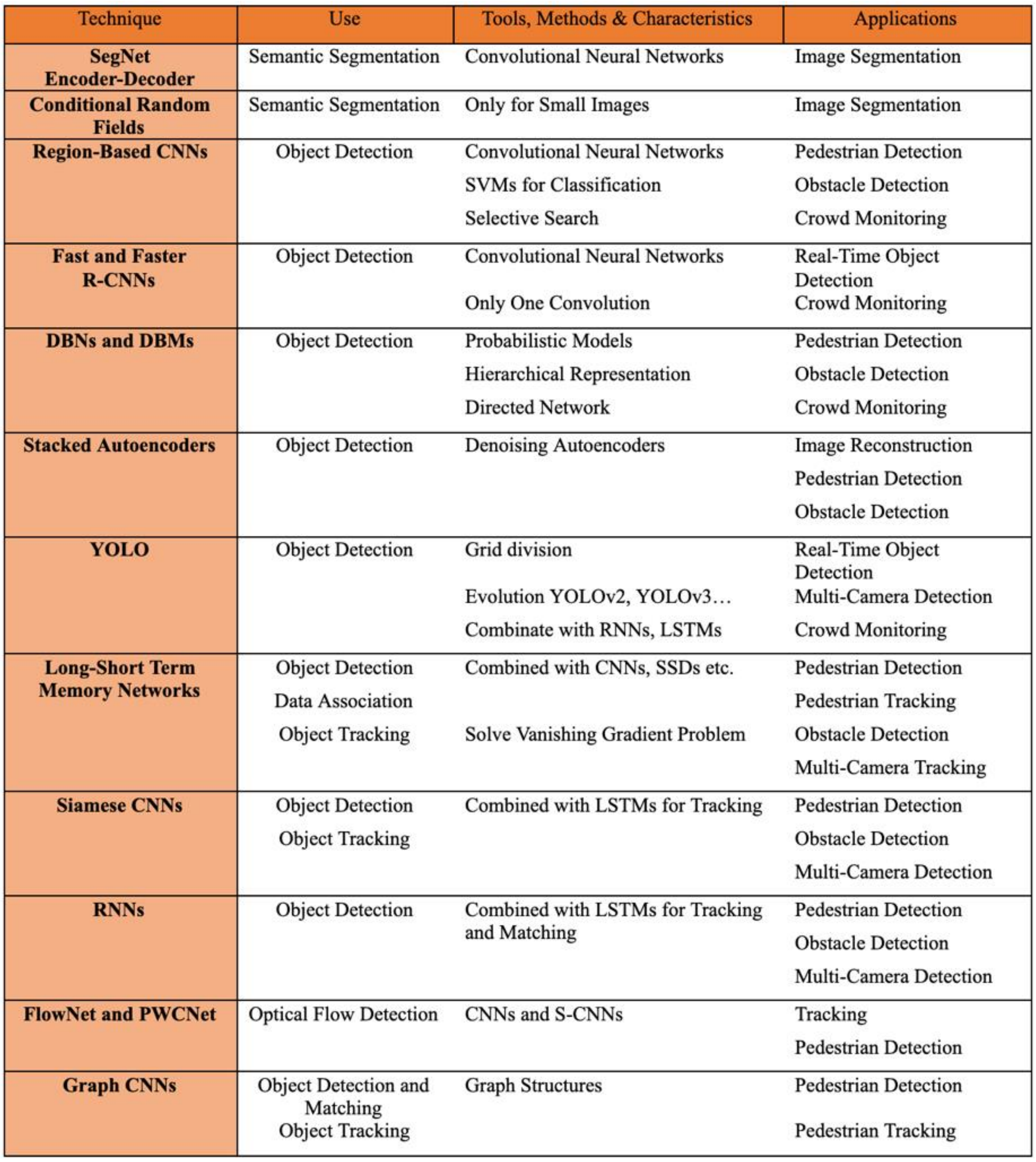
3.5 SUMMARY OF MODERN AND TRADITIONAL TECHNIQUES

In this subsection, a summary of both traditional techniques and modern techniques will be presented for the tasks that each technique has proven useful.



Technique	Use	Tools, Methods & Characteristics	Applications
Image Processing	Object Detection	Background Subtraction Histogram EQ, Contrast & Intensity Adjustments	Video Motion Detection Multi-Person tracking
Nearest Neighbors	Object Recognition	Making People-shaped Blobs	Video Motion Detection Multi-Person tracking
Frame correspondence	Object Tracking	Feature Vectors	Multi-Person tracking
Neural Networks	Object Detection Classification	Feed Forward Neural Networks Neuron classifiers	Video Motion Detection Multi-Person tracking
Kalman Filtering	Tracking by prediction	Combination with P2DHMM	Multi-Person tracking
P2D Hidden Markov Models	Object Detection	HMM Multi-Stream Laplace of Gaussian Filtering	Multi-Person tracking
Support Vector Machines	Texture Analysis	Image Pre-Processing Density Cell Model Feature Extraction	Crowd Density Abnormal Density detection
Self-Organizing Maps	Texture Analysis	Kohonen maps GLDM, Straight Line Segmentation and Fourier Spectrum	Crowd Monitoring
Bayesian Poisson Regression	Object Detection Texture Analysis	Approximate Poisson regression to Gaussian Processes GLDM, Straight Line Segmentation and Fourier Spectrum	Crowd Counting Crowd Monitoring
Polynomial Fitting Functions	Classification	Estimate coefficients from training samples	Crowd Monitoring

Table 1: Summary of Traditional Techniques for Human Detection and Tracking



Technique	Use	Tools, Methods & Characteristics	Applications
SegNet Encoder-Decoder	Semantic Segmentation	Convolutional Neural Networks	Image Segmentation
Conditional Random Fields	Semantic Segmentation	Only for Small Images	Image Segmentation
Region-Based CNNs	Object Detection	Convolutional Neural Networks SVMs for Classification Selective Search	Pedestrian Detection Obstacle Detection Crowd Monitoring
Fast and Faster R-CNNs	Object Detection	Convolutional Neural Networks Only One Convolution	Real-Time Object Detection Crowd Monitoring
DBNs and DBMs	Object Detection	Probabilistic Models Hierarchical Representation Directed Network	Pedestrian Detection Obstacle Detection Crowd Monitoring
Stacked Autoencoders	Object Detection	Denoising Autoencoders	Image Reconstruction Pedestrian Detection Obstacle Detection
YOLO	Object Detection	Grid division Evolution YOLOv2, YOLOv3… Combinate with RNNs, LSTMs	Real-Time Object Detection Multi-Camera Detection Crowd Monitoring
Long-Short Term Memory Networks	Object Detection Data Association Object Tracking	Combined with CNNs, SSDs etc. Solve Vanishing Gradient Problem	Pedestrian Detection Pedestrian Tracking Obstacle Detection Multi-Camera Tracking
Siamese CNNs	Object Detection Object Tracking	Combined with LSTMs for Tracking	Pedestrian Detection Obstacle Detection Multi-Camera Detection
RNNs	Object Detection	Combined with LSTMs for Tracking and Matching	Pedestrian Detection Obstacle Detection Multi-Camera Detection
FlowNet and PWCNet	Optical Flow Detection	CNNs and S-CNNs	Tracking Pedestrian Detection
Graph CNNs	Object Detection and Matching Object Tracking	Graph Structures	Pedestrian Detection Pedestrian Tracking

Table 2: Summary of Modern Techniques for Human Detection and Tracking

3.6 RESEARCH AND USE OF CV IN HIGH TECH COMPANIES

Since this area of investigation is as innovative as it gets, top tech companies have invested in research and development in computer vision over the last few years.

Google of course is one of the companies that invested more on the field. They have a full team in charge of building systems that interpret and transform sensory data. Within this sensory data, image understanding is one of the prime fields of investigation. Real-time video understanding is also one of the fields that teams investigate, taking into consideration that Google owns Youtube, video and motion skills is one of their principal fields of research. A lot of applications regarding Google Photos and Image Search have been developed to further the research into the field. One of the most known architectures developed by Google in image understanding and in combination with a deep learning neural network is called Inception. This architecture processes several convolutions in parallel with a max pooling layer to later do filter concatenation. This architecture proves that it improves the neural network performance by approximating the sparse structure with readily available building blocks. This allows the network to increase its quality significantly and just with a slight increase of computational cost. Not only research, but Google also offers in their GCP a computer vision platform called Cloud Vision.

Google has always proved a reliable collaborator to the research community, not only their private affairs. In our case of human detection specifically, they have shown progress also.
In the next image, their result in the public COCO challenge was so promising that they implemented their model in their own product NestCam, which is a surveillance camera application for the home [21], and in the well-known Google Street View. They also made TensorFlow Object Detection API available for the public, which include several trainable models like SSDs, also in combination with Inception, Faster CNNs and Region-based CNNs.

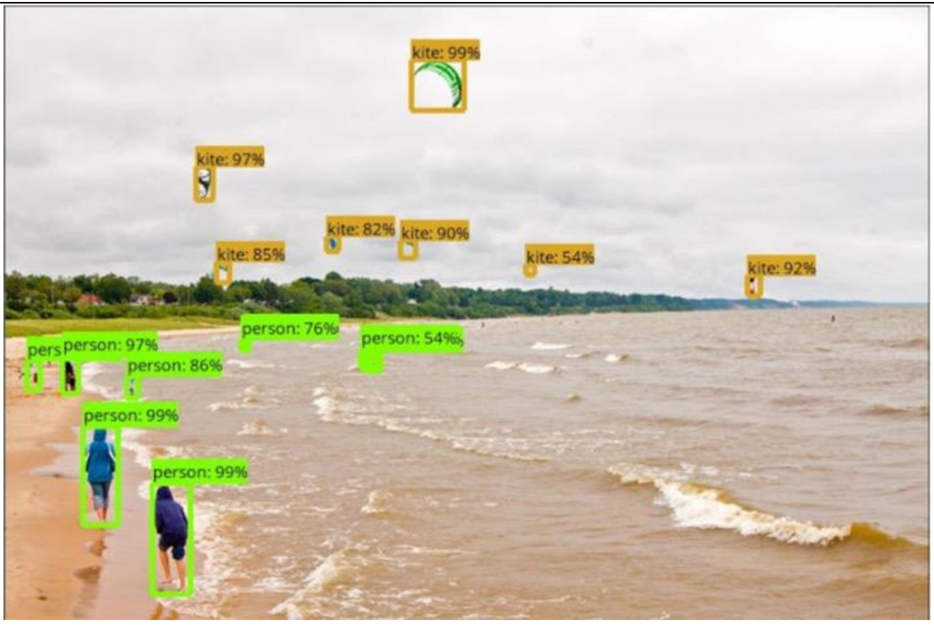


Figure 37. Google Model for Detection

Amazon is one of the main companies also, they have done a lot of research and development of their own in the field. Amazon Rekognition is a platform developed by Amazon that allows to integrate deep learning and computer vision together without any knowledge of machine learning whatsoever. Identification of objects, people and activities in images and videos is made easy with this platform. This platform allows you to build a model to the user needs and you only are responsible for providing the image and videos.

The key features of this platform are all that someone could imagine from a computer vision platform. Labeling both generally and specifically of images and videos is possible, even in real-time. Face detection and verification is another of its features and finally tracking. Companies like the NFL are known to use Amazon Rekognition for the analysis of their games, player paths, player identifications etc. CBS television corporation is another company that uses Amazon Rekognition.

This platform could be very useful as an aspiration for our project in terms of an end-to-end application for human detection, tracking and notifying the distance constraints.

To end this brief review of computer vision in big companies, Nvidia is one of the companies that has done extensive research and development.

Nvidia has an extensive publication library of papers, research and development in the computer vision area. In the majority of them, the approaches used are the ones described in the previous sections. One of them compares the performance of RNNs and Bayesian filters (Kalman filters in particular) for tracking and facial detection. The methods they proposed are shown in the next image. As seen, CNNs are still the keystone of the models. As the paper shows, the end-to-end RNNs approach yields better results for face detection.

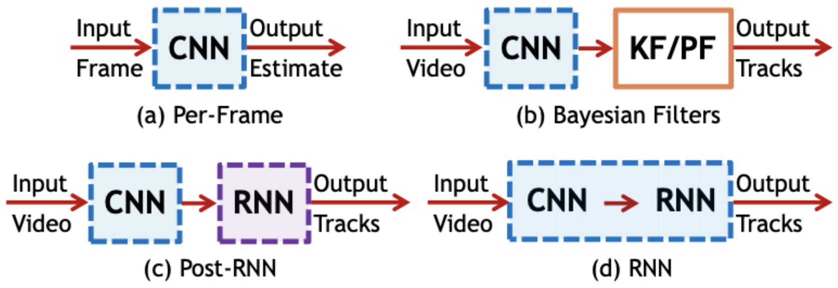


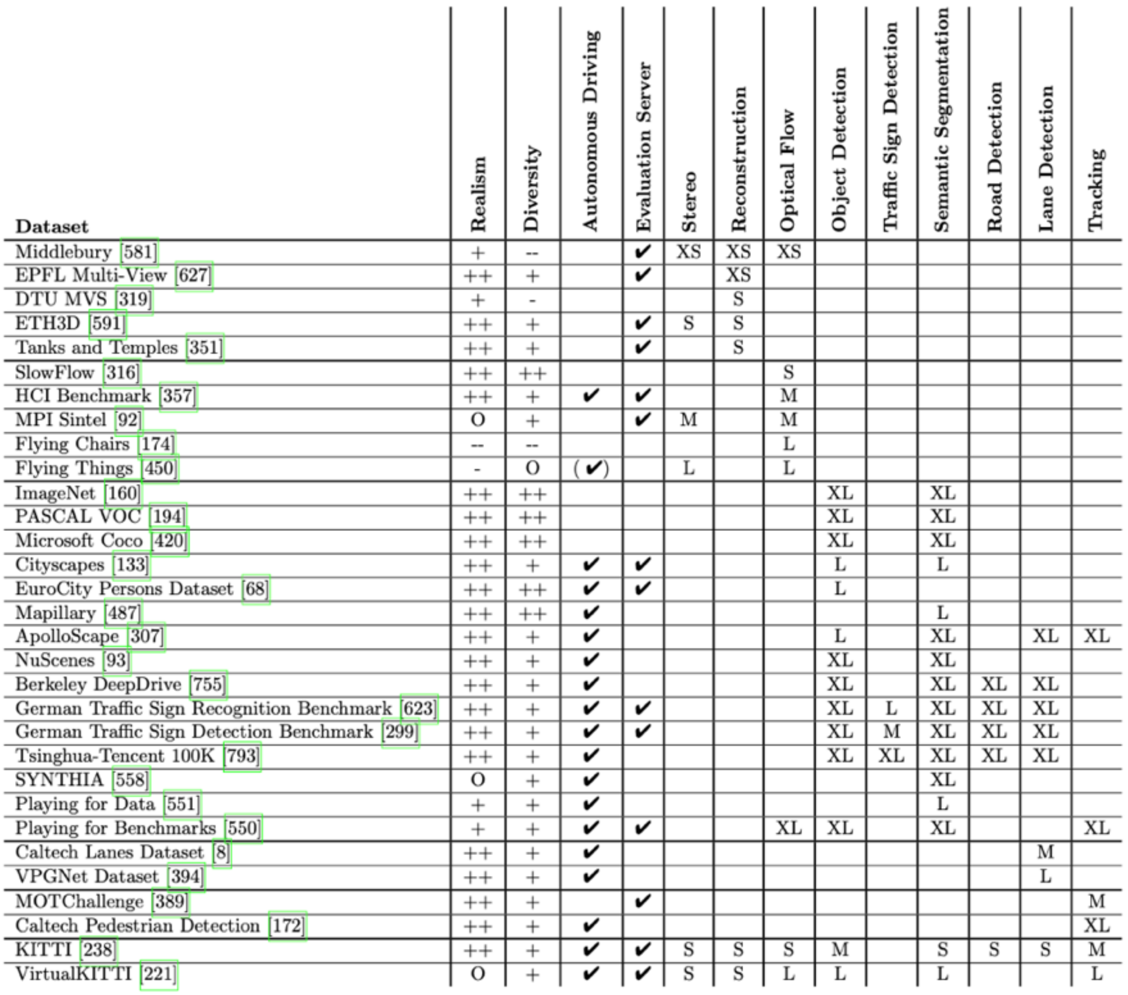
Figure 38. Models Employed by Nvidia [31]

Other companies that provide end-to-end APIs and platforms are for example Microsoft, IBM with IBM Watson, which is a very complete and used platform for cloud computing.

3.7 DATASETS

Since the start of the machine learning, deep learning boom, the thing that was the key to all of it wasn´t and still isn´t the models, techniques or tools. The key to all advances and developments has been the data, how it is structured, presented and gathered. For this reason, using the right datasets for a project is probably the first important decision to take. In this section, different options of suitable datasets for our case will be presented. From only a few years ago to nowadays, the dataset size for reliable training and testing has grown from just a few hundred of examples to the thousands and hundreds of thousands of examples.

From [15], a great comparison of several datasets for autonomous driving, which include pedestrian detection and tracking tasks is presented, and I will use it to select possible datasets useful for our objective.



Dataset	Realism	Diversity	Autonomous Driving	Evaluation Server	Stereo	Reconstruction	Optical Flow	Object Detection	Traffic Sign Detection	Semantic Segmentation	Road Detection	Lane Detection	Tracking
Middlebury [581]	+	--		✔	XS	XS	XS						
EPFL Multi-View [627]	++	+		✔		XS							
DTU MVS [319]	+	-				S							
ETH3D [591]	++	+		✔	S	S							
Tanks and Temples [351]	++	+		✔		S							
SlowFlow [316]	++	++					S						
HCI Benchmark [357]	++	+	✔	✔			M						
MPI Sintel [92]	O	+		✔	M		M						
Flying Chairs [174]	--	--					L						
Flying Things [450]	-	O	(✔)		L		L						
ImageNet [160]	++	++						XL		XL			
PASCAL VOC [194]	++	++						XL		XL			
Microsoft Coco [420]	++	++						XL		XL			
Cityscapes [133]	++	+	✔	✔				L		L			
EuroCity Persons Dataset [68]	++	++	✔	✔				L					
Mapillary [487]	++	++	✔							L			
ApolloScape [307]	++	+	✔					L		XL		XL	XL
NuScenes [93]	++	+	✔					XL		XL			
Berkeley DeepDrive [755]	++	+	✔					XL		XL	XL	XL	
German Traffic Sign Recognition Benchmark [623]	++	+	✔	✔				XL	L	XL	XL	XL	
German Traffic Sign Detection Benchmark [299]	++	+	✔	✔				XL	M	XL	XL	XL	
Tsinghua-Tencent 100K [793]	++	+	✔					XL	XL	XL	XL	XL	
SYNTHIA [558]	O	+	✔							XL			
Playing for Data [551]	+	+	✔							L			
Playing for Benchmarks [550]	+	+	✔	✔			XL	XL		XL			XL
Caltech Lanes Dataset [8]	++	+	✔									M	
VPGNet Dataset [394]	++	+	✔									L	
MOTChallenge [389]	++	+		✔									M
Caltech Pedestrian Detection [172]	++	+	✔										XL
KITTI [238]	++	+	✔	✔	S	S	S	M		S	S	S	M
VirtualKITTI [221]	O	+	✔	✔	S	S	L	L		L			L

Figure 39. Dataset Valuation [15]

For object recognition, ImageNet, PASCAL VOC and Microsoft's COCO are three datasets widely known for training and testing models with this task. PASCAL VOC (Virtual Object Classes) was the dataset for an EU funded challenge consisting in object classification, detection and segmentation based on photographs collected from users of Flickr. COCO was a Microsoft dataset design for object detection and image segmentation for the further understanding of an image. COCO is significantly larger than PASCAL (328k images in contrast to 11530). To this date, these three datasets are the most complete and reliable datasets for object detection. A sample of each dataset is presented next.

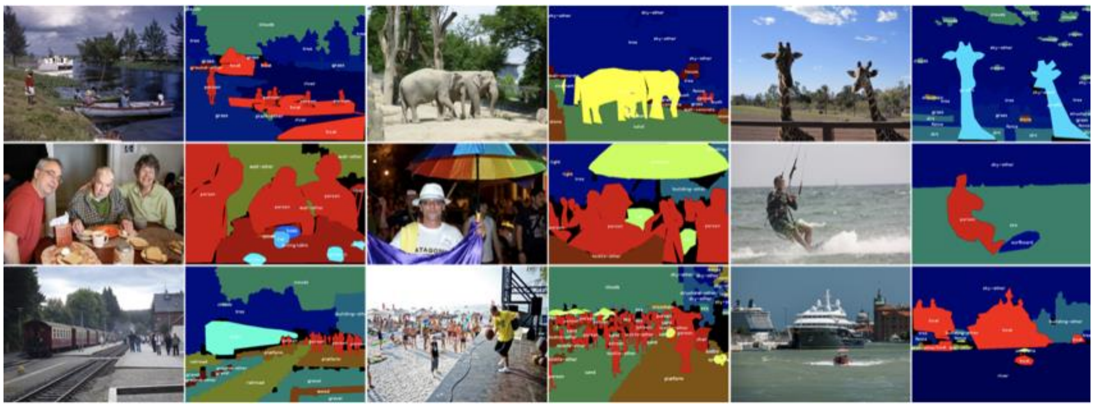


***Figure 40. COCO*[36]**

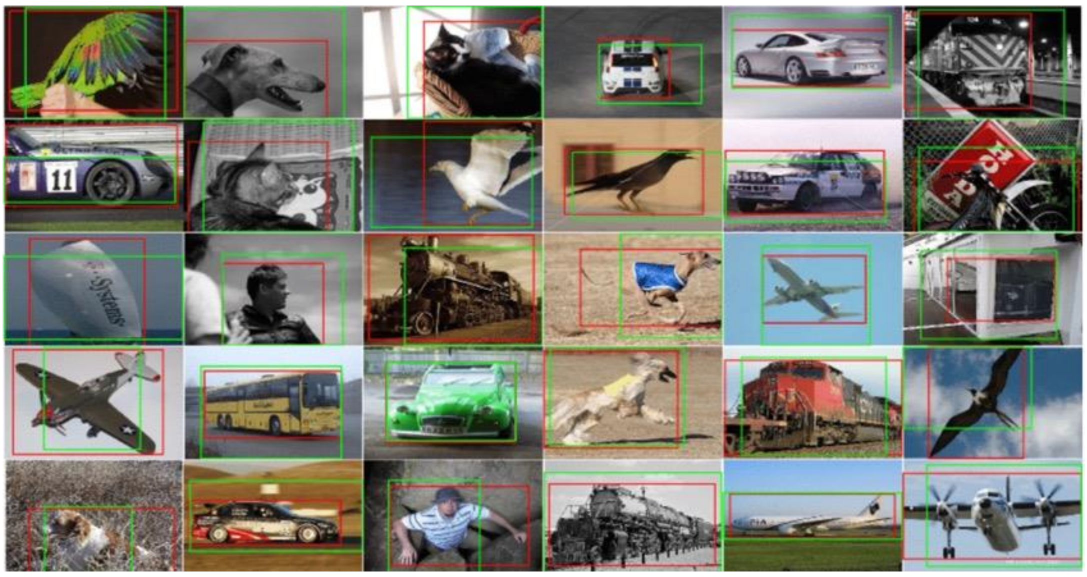


***Figure 41. PASCAL VOC*[37]**

36 https://github.com/nightrome/cocostuff

37 https://www.researchgate.net/figure/Results-of-localization-on-PASCAL-VOC-dataset-17-Green-box-Estimated-Window-Red_fig3_318029536

In terms of tracking, the MOT Challenge has been widely used. It has 14 video sequences filmed with static and video cameras. The challenge combines several benchmarks such as KITTI, another widely known dataset for detection and tracking. KITTI is designed for the autonomous driving context, but it could be really useful for us also for pedestrian detection. The usability of KITTI in pedestrian detection and tracking can be seen in the following two images.

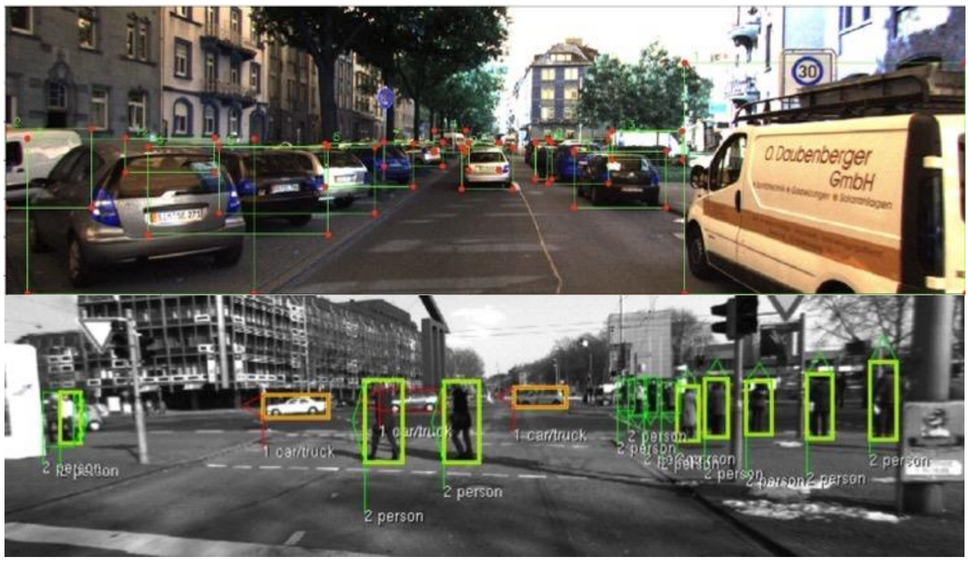


Figure 42. KITTI[38]

Specifically, for our case, after analyzing all the datasets mentioned in [15], there are two more, apart from the previously mentioned that fit our needs almost perfectly. Mapillary Vistas is a dataset of street view images. It self-claimed the largest and most diverse Street-level dataset and it is, at least the largest, with more than 130 million images from all around the globe. The next image shows already segmented images from the dataset.

38 http://www.cvlibs.net/datasets/karlsruhe_objects/

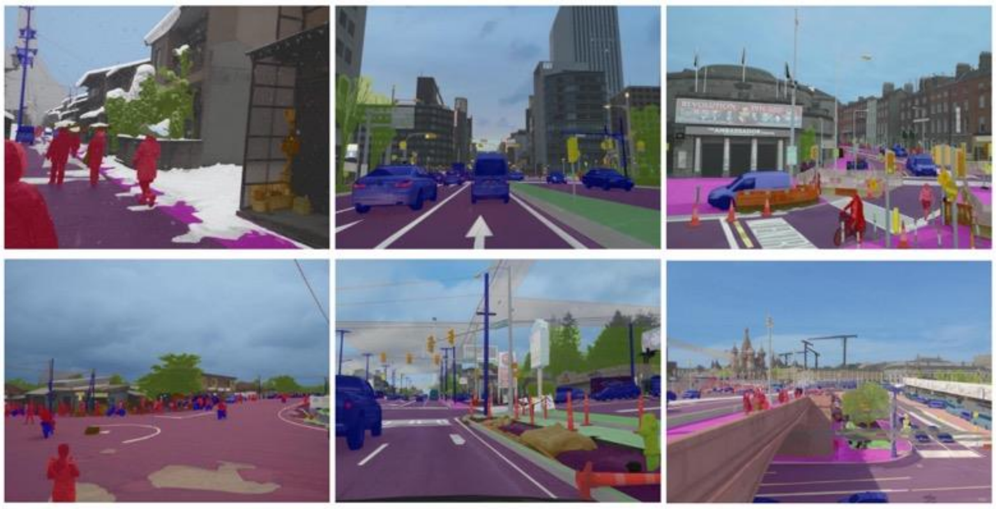


Figure 43. Mapillary Vistas Dataset[39]

The same developers have created a Street-Level Sequences dataset of over 1.6 million images in a large number of short sequences that also could be used for tracking.

The Caltech Pedestrian Detection benchmark might be, out of all of these datasets, the most appropriate for our case. It has 10 hours of video of street level pedestrians, as it can be seen in the next image.

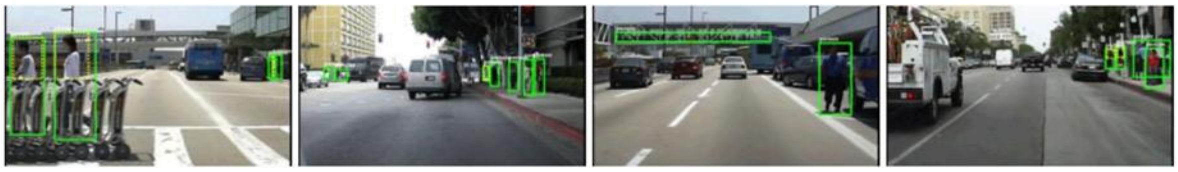


Figure 44. Caltech Pedestrian Dataset Example[40]

39 https://blog.mapillary.com/product/2017/05/03/mapillary-vistas-dataset.html

40 http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/

CHAPTER 4. SCOPE OF THE PROJECT

4.1 JUSTIFICATION

In light of the previous work on similar problems and the state of the art on these technologies and techniques, I think that being able to provide this autonomous monitorization of interpersonal distance could be a huge advance both in the current situation and in the future, since it can be used to control and monitor crowds in different scenarios. As seen, there are methods that measure distance between objects or individuals, the difference from this project is that most of those projects measure distance by taking references from the scene and then using them as thresholds, measuring the distance in a traditional way against those thresholds. Those methods are unique for each perspective of the camera and the scenario. What we want is to estimate the distance, meaning that, by using Deep Learning, we estimate it without taking any thresholds or references, making the system completely independent of the perspective and scenario.

4.2 USE OF STATE-OF-THE-ART METHODS AND TECHNOLOGIES

For this project, the deep learning models used for detection and tracking are state of the art methods such as YOLO or DeepSORT, very recent developments and very useful for our task in hand.

4.3 DISTANCE ESTIMATION

As mentioned before, it is true that the estimation of distance is an area of investigation very much researched, but I believe that the approach we are going to try in this project is not that common. It will be based in the depth information that certain cameras can get and rely on the power of DNNs and CNNs to analyze that information and provide an accurate estimation of the distance between individuals.

4.4 REAL-TIME OPERATION

Of course, any project that tries to monitor something in a video needs to be able to process data and work in real-time. The definition of real time is wide, but for this project I am going to define the limit of real-time processing to be 30 Frames Per Second, which is a very common estimation. A lot of applications can be benefited if we manage to reach real time processing in this project. It is not an easy thing to do and I will need to carefully choose models and techniques that provide that speed and be sufficiently accurate to be able to detect and track with precision.

4.5 OBJECTIVES

- The first objective is clear and already defined: Determine if it is possible to estimate the distance between objects, individuals etc. without depending on references from the scene, perspective from the cameras and previousl fixed thresholds.
- The second objective is to provide a system that is able to work in real-time. As mentioned before, any processing speed we reach higher than 30 FPS will be considered as real-time processing.
- The third and final objective is to provide an accuracy high enough to be sure that individuals can be detected easily in clear situations. Obviously working with occlusions and defective images is more difficult but the accuracy must be very good in clear scenarios.

4.6 METHODS USED

To reach the objectives, I am going to build a detector based probably on YOLO or a similar model, since it is the state-of-the-art on real time processing detection algorithms. The result from the algorithm will be fed into a tracking algorithm to help with dealing with occlusions and trajectories and finally, I will develop a model whose objective is to determine if we can estimate the distance between individuals. If I have enough time, I will try to chain together the three stages, but first, they must work individually within our objectives of real time and accuracy.

CHAPTER 5. DETECTORS

5.1 INTRODUCTION TO OBJECT DETECTION ALGORITHMS

As mentioned thoroughly through all this document, in all things related to Deep Learning, AI and Computer Vision there has been and always will be both a traditional and a modern approach.

To refresh our memories of what traditional and modern techniques do, let’s provide a brief description and some key characteristics about them. Traditional approaches for object detection normally have three phases. The first one is the informative region selection. In this phase the objective is to determine the object location in the frame. Since objects have different sizes and shapes and can appear at different locations. To adapt to this situation, a multiscale sliding window is used, but it is computationally expensive. The second phase is to do the feature extraction. As mentioned earlier, this consists in extracting visual features to make a semantic and robust image representation. Two techniques widely used for this in the past are SIFT[54] (Space Invariant Feature Transform) and HOG. This phase is very important but also very difficult due to occlusions, different illuminations and backgrounds. The final phase is to do the classification, which is normally done using SVMs or AdaBoost algorithms. As expected, this traditional approach is computationally very expensive and provides worse results than the following modern approach.

The exponential growth of Deep Learning is what allowed us to have a modern approach. As outlined before, it allows us to extract more complex features of the images, getting a better representation of the feature space. There are basically two frameworks where all the previous techniques exposed are grouped in. These frameworks are region proposal algorithms, such as R-CNNs and the other framework consists in regression-based algorithms, such as YOLO (and all of its versions). As mentioned earlier, R-CNN architecture generates the proposals of objects bounding boxes first, then does the feature extraction and finally does the classification and localization. As main drawbacks, doing a selective search of locations is very expensive computationally and takes more time. However, it´s accuracy is very good, reaching a mAP of almost 55%. As seen earlier, there are architectures such as Fast-RCNN and Faster-RCNN which can provide a faster training, but it is still considerably long. Essentially, these approaches are not very suitable for real-time applications, just by how they are built.

The regression/classification-based frameworks are different because they search for spaces in an image with high probabilities of containing an object. The main algorithm within this approach is YOLO (You Only Look Once) and its different versions. Because of the detection only being done in one step, this algorithm is considerably faster than the different

RCNNs algorithms. However, it's accuracy is a little bit lower. This is because it usually struggles with small objects in groups which causes the accuracy to drop.

The common use cases for all these detection algorithms are already mentioned in this document, but since these two different frameworks of algorithms have different advantages and limitations, there are certain applications that are more suitable to do using RCNN-based algorithms and other applications that are more suitable to perform better when using regression-based algorithms such as YOLO.

By general rule, all applications where real-time processing is key to their performance should be implemented with regression-based algorithms because of the improved speed they provide. Common tasks for this framework are autonomous vehicles. In this case it is very clear the importance of providing a real-time processing with the smallest delay possible. Other types of tasks are tracking tasks such as this project, where we want to track and monitor movement of people. A similar task is the monitoring of wildlife or different military applications using drones to control movement.

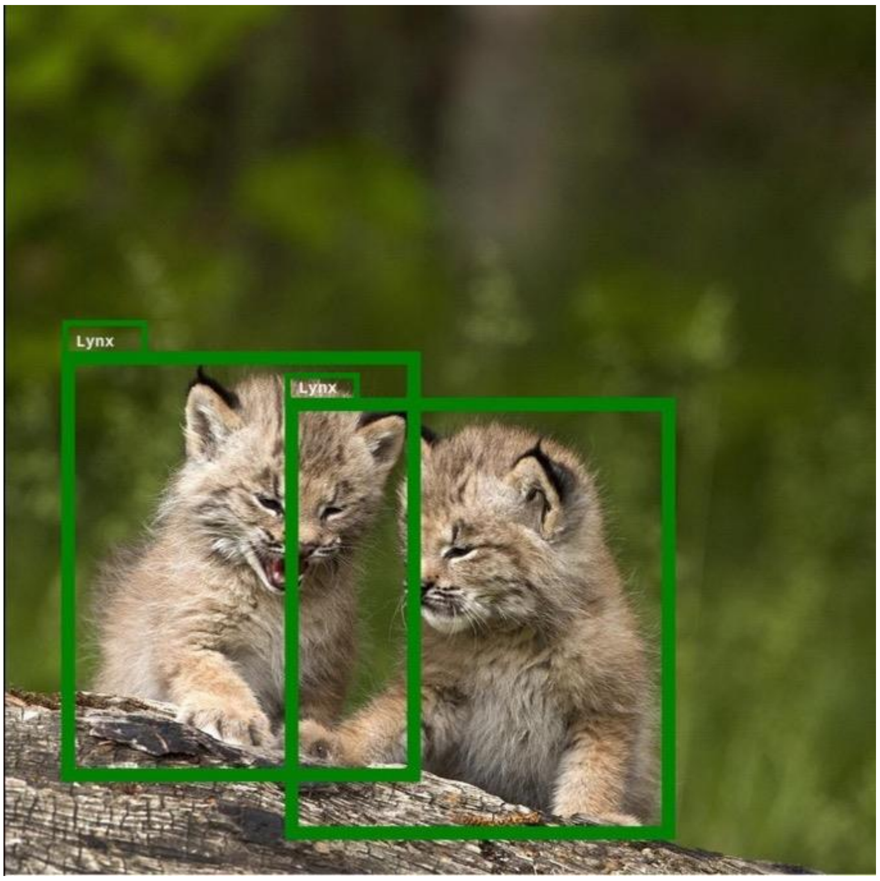


Figure 45. Wildlife Monitoring Using YOLO [41]

For the case of RCNN-based frameworks, applications where these algorithms are very useful are, by contrast, ones more “static”, where the real-time constraint is not a key performance indicator. Use cases where this has become useful are the following: The analysis of the underwater background to detect anomalies, facial recognition and medical examinations. These are clear examples where the higher accuracy of these frameworks is a more important parameter.

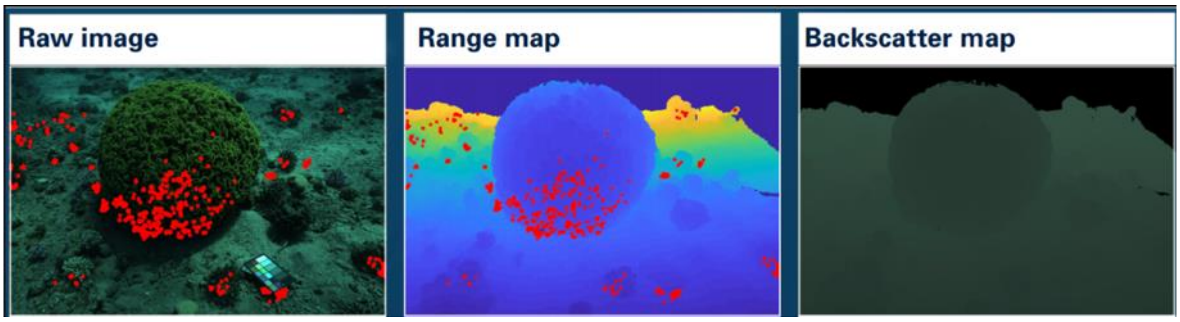


Figure 46. Underwater anomalies detected by an RCNN framework

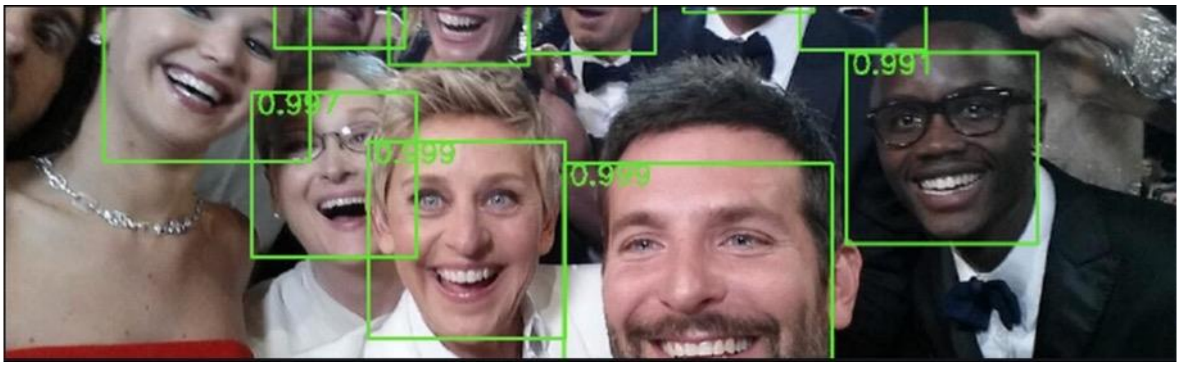


Figure 47. Facial Recognition using an RCNN framework

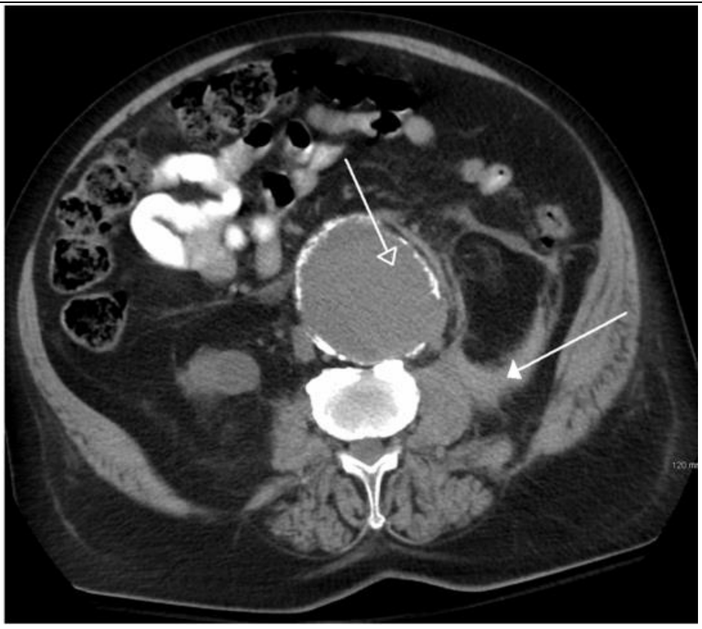


Figure 48. Cerebral Examination using RCNN architectures [42]

As explained earlier, taking into consideration the limitations and the advantages of all these different algorithms and the different use cases for each of them, it is fairly logical that the optimal algorithm to use for our people detection in real-time use case is YOLO and its different versions.

In the following sections I will compare the last three releases of YOLO (YOLOv3, YOLOv4 and YOLOv5) both in structure and results, training them on three different datasets such as COCO, KITTI and CrowdHuman, which are, apriori, suitable datasets for our use case.

5.2 DATASETS TO BE USED

As shown earlier, there is an enormous number of different datasets available to perform training and modelling with them. Since the objective of this detection phase is to determine which of these datasets is the best one for our task, some of them have been trained with the different YOLO versions and looking at the results from each of them, which will be outlined in a later section, a decision is going to be made whether one, some of none of the chosen datasets is appropriate for our task. The "winner" will be the one used for the models fed into the tracking and the estimation of distance. Initially, the datasets chosen to be tried with the different frameworks are COCO (Common Objects in COntext), because it is the unofficial benchmark for object detection, KITTI, another widely used dataset for object detection that, as said earlier, is commonly used for a real-time application like autonomous vehicles and it could be useful to translate it to our problem. The third one is the Caltech pedestrian dataset. With this dataset, with similar objectives as KITTI, the results obtained (shown later) are not good and this dataset will be discarded and substituted by the CrowdHuman dataset. The bad results of the Caltech dataset is because it does not have enough human labels to train with and the resolution is very poor, compared to a similar dataset like KITTI, but this will be explained later more clearly. CrowdHuman is a dataset not explained before that actually might be the best option to train our models with. It is a dataset of internet images annotated only with person and head labels. It is fairly recent and not commonly used yet, but it is the most accurate for this problem.

With these datasets, the models will be trained on datasets of very different size and content, challenging them to generalize good enough and be reasonably accurate to feed them into the tracking phase.

Before diving deep into the models, their architectures, and results, I am going to provide some light into the formats and how the datasets work. To be efficient, I want to use only a script to load the datasets to the models. To be able to do this, some formatting and reorganization of directories of the different datasets is needed. **You can find the script and all the scripts mentioned in this section in Annex I.**

5.2.1 COCO

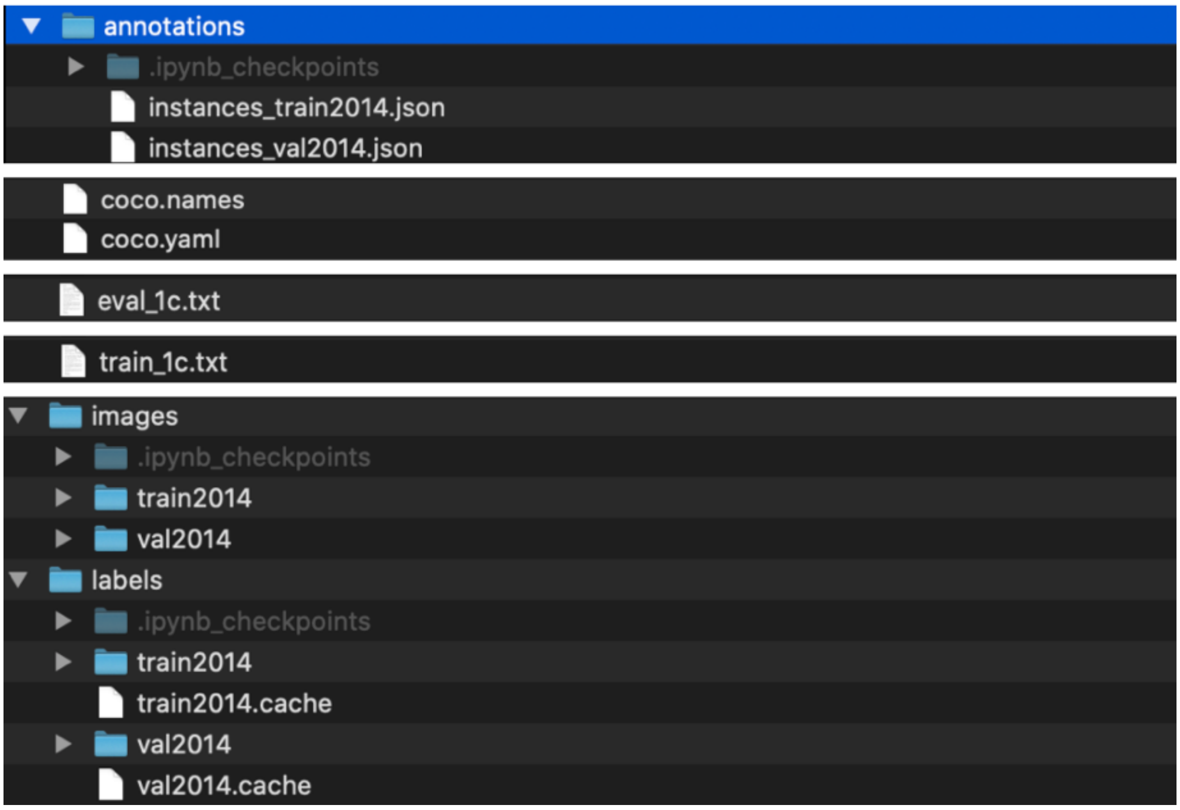
COCO as described earlier is the most used dataset for object detection problems. It is very large and very complete. Furthermore, it is so complete that they have an available API to allow the users to download the images and annotations through a Python script and a connection to the API (It is also available for Matlab).

This has been very useful because for our case, the only images we actually need are the ones containing people. Running the script called **coco_persons.py** that is in the **Annex I**, I was able to download the images and the annotations of only the images with people in it. Unfortunately, some images were corrupted so I had to download the whole dataset manually and use another script (**copy_images.py**) to correlate the necessary images to the previously downloaded annotations so only images with people are used.

COCO's directory structure is as follows, and I am going to explain it because I will use that structure for the other datasets as well. COCO has annotation files in JSON format with the information regarding the images. This includes bounding boxes coordinates, image IDs, labels etc. Then it has a directory for images, with subdirectories for the training and validation images and at the same level as the images directories there is a labels directory, as well as the corresponding subdirectories for training and validation.

This is very similar to the format YOLO uses which has a slight difference. YOLO has a specific labelling format. Each image has a .txt file associated with the labels and coordinates of the bounding boxes and the algorithm will look for that file when processing the image.

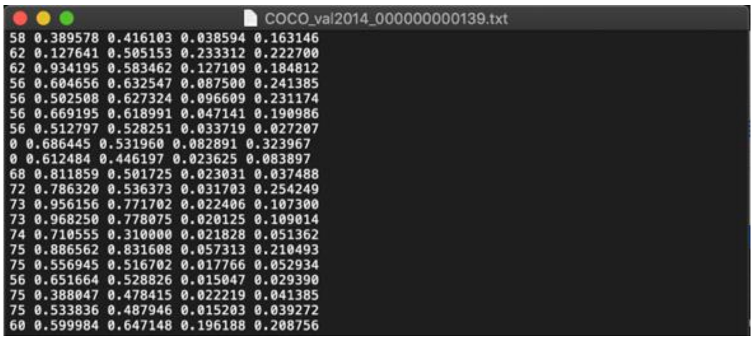
This means that it is necessary to convert the JSON files to separate .txt files for the correct processing in YOLO. All in all, the structure ends up something like the following:



```
▼ annotations
    ▶ .ipynb_checkpoints
      instances_train2014.json
      instances_val2014.json
  coco.names
  coco.yaml
  eval_1c.txt
  train_1c.txt
▼ images
    ▶ .ipynb_checkpoints
    ▶ train2014
    ▶ val2014
▼ labels
    ▶ .ipynb_checkpoints
    ▶ train2014
      train2014.cache
    ▶ val2014
      val2014.cache
```

Figure 49. Directory Structure for COCO (and all) dataset

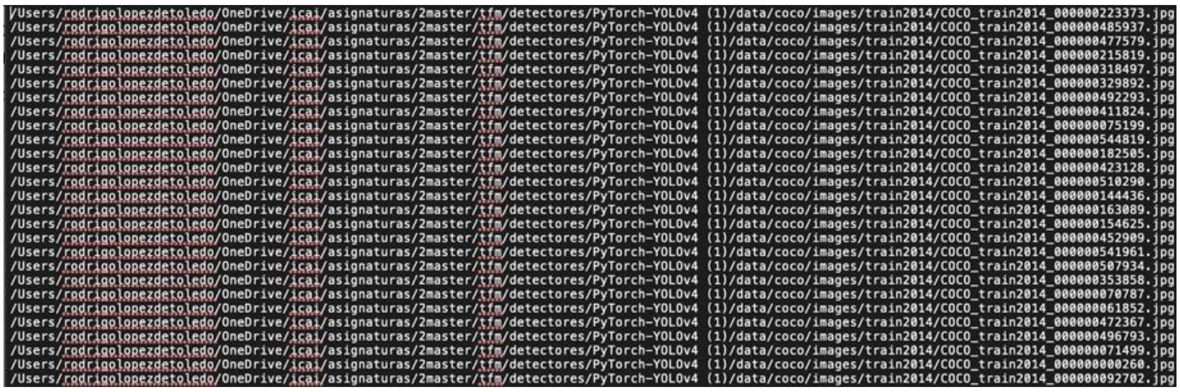
Apart from the directory structure explained before, there are some files used in this. The first one is a YAML file with the directories of the training and validation images, the number of different classes and the names of those classes. Then there are two TXT files, one for training and one for validation with the paths of all the images to use be used for training. Using auxiliary scripts like (**reduce_dt.py**) the number of images can be selected. To generate these files, use the script (**get_text_file.py**).



COCO_val2014_000000000139.txt

```
58 0.389578 0.416103 0.038594 0.163146
62 0.127641 0.505153 0.233312 0.222700
62 0.934195 0.583462 0.127109 0.184812
56 0.604656 0.632547 0.087500 0.241385
56 0.502508 0.627324 0.096609 0.231174
56 0.669195 0.618991 0.047141 0.190986
56 0.512797 0.528251 0.033719 0.027207
0 0.686445 0.531960 0.082891 0.323967
0 0.612484 0.446197 0.023625 0.083897
68 0.811859 0.501725 0.023031 0.037488
72 0.786320 0.536373 0.031703 0.254249
73 0.956156 0.771702 0.022406 0.107300
73 0.968250 0.778075 0.020125 0.109014
74 0.710555 0.310000 0.021828 0.051362
75 0.886562 0.831608 0.057313 0.210493
75 0.556945 0.516702 0.017766 0.052934
56 0.651664 0.528826 0.015047 0.029390
75 0.388047 0.478415 0.022219 0.041385
75 0.533836 0.487946 0.015203 0.039272
60 0.599984 0.647148 0.196188 0.208756
```

Figure 50. Example of .TXT annotation file for image 000000000139 of the validation dataset



```
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000223373.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000485937.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000477579.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000215819.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000318497.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000329892.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000492293.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000411824.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000075199.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000544819.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000182505.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000423128.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000510290.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000144436.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000163089.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000154625.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000452909.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000541961.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000507934.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000353858.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000070787.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000061852.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000472367.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000496793.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000071499.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000000260.jpg
/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyTorch-YOLOv4 (1)/data/coco/images/train2014/COCO_train2014_000000092702.jpg
```

Figure 51. File with the paths of all the images for training

All these files are used when loading the dataset into the model. The pipeline is as follows. The main script processes the YAML file and retrieves the paths of the file with the paths of all the images and the class names. In a for loop, it processes that file and get the image and to get the annotations, it substitutes from the path “images” for “labels” and, since the rest of the path is exactly the same, from that image path, it can locate the TXT file with its corresponding annotations. All images and annotations are stored in a PyTorch dataloader to use for training.

This structure and pipeline will be the same for all datasets.

5.2.2 KITTI 3D

The transformation from KITTI is easy. The only difference is that it has a directory for training with images and labels and a directory for validation. The only change was to move all the images to an image directory and all the labels to a label’s directory.

In the TXT files for each image, the difference is that the class is specified in plain text while YOLO is in a number format (Person is id=0). After translating those labels, the pipeline works as needed.

5.2.3 CALTECH PEDESTRIAN DATASET

Transforming the Caltech dataset was much more complicated. The image, since this dataset is sequence videos, were stored as .seq files and the annotations as .vbb files.

VBB format is a matrix format, there is a matrix for each image so what it was needed to do was to convert each row of the matrix into a row in a text file. This is done in **generate_annot.py.**

For the SEQ files, OpenCV allows opening this type of images, so the script **generate_imgs.py** opens the files with OpenCV and saves them as .png.

However, the images of these dataset are way out of YOLO image size. They are very wide and not high enough, so they needed to be squared somehow. That is probably the reason the results were so poor when using this dataset, as it is shown later.

5.2.4 CROWDHUMAN

Finally, with the CrowdHuman dataset, the directory structure was fine but the annotations are given in *.odgt* format. Not only that, but the coordinates were in a different format than what YOLO uses. I needed to normalize them with the image size that the model uses (640x640) and save each row of the ODGT file to a row of a TXT file with these new values. This is done in the **train_annot.py** and **val_annot.py** files.

After all these transformations all datasets can be processed the same way and if there is one mistake, fixing it for one means fixing it for all of them.

5.3 YOLO

5.3.1 ¿WHY YOLO?

The main reason is outlined above. However, it is important to dig a little deeper into why this algorithm is as good as they say when dealing with real-time problems. YOLO means You Only Look Once, meaning that the algorithm passes through the image completely in a single run. By doing this, it is able to process 155 frames per second, a much higher rate than RCNN solutions, which vary between 5 and 20 frames per second, as outlined before and in the bibliography.

The algorithm divides the input image into a grid, and, within this grid, it generates the bounding boxes and the class probabilities of those boxes. After generating all the possible boxes for the input image, which is done by passing it through a Fully Convolutional Neural Network, the information of class probabilities and boxes coordinates is passed through a Non-Max Suppression algorithm that eliminates the bounding boxes containing none or very little information (Usually those with very little class probability) and only keeps the bounding box more accurate for each object (or localization in the input image).

It is important to analyze the drawbacks of this algorithm and why, in spite of them, it is a good option for this project.

Mentioned earlier is the slight drop of accuracy when using this algorithm instead of a region-proposal based algorithm. It is not much but it is important to know that it is highly improbable to reach maximum accuracy with one of these algorithms. The accuracy will always be slightly lower than RCNN algorithms such as Detectron2 (Facebook Research) [40]**.**

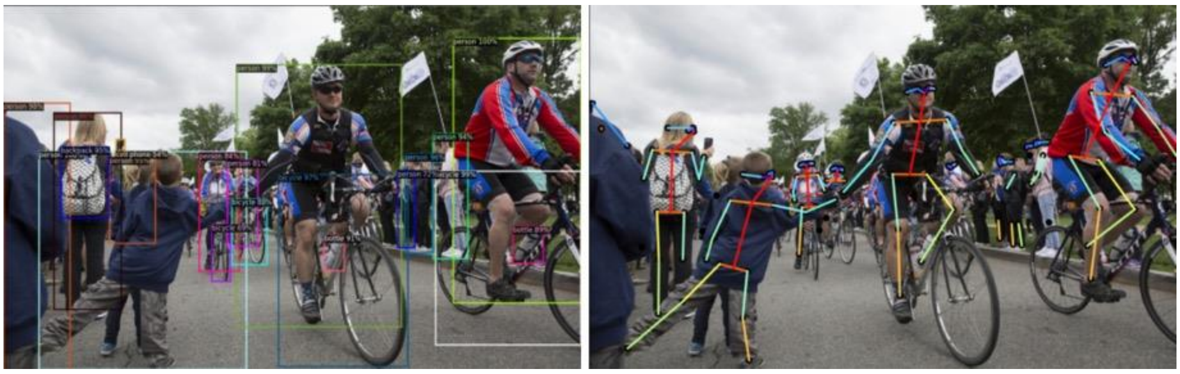


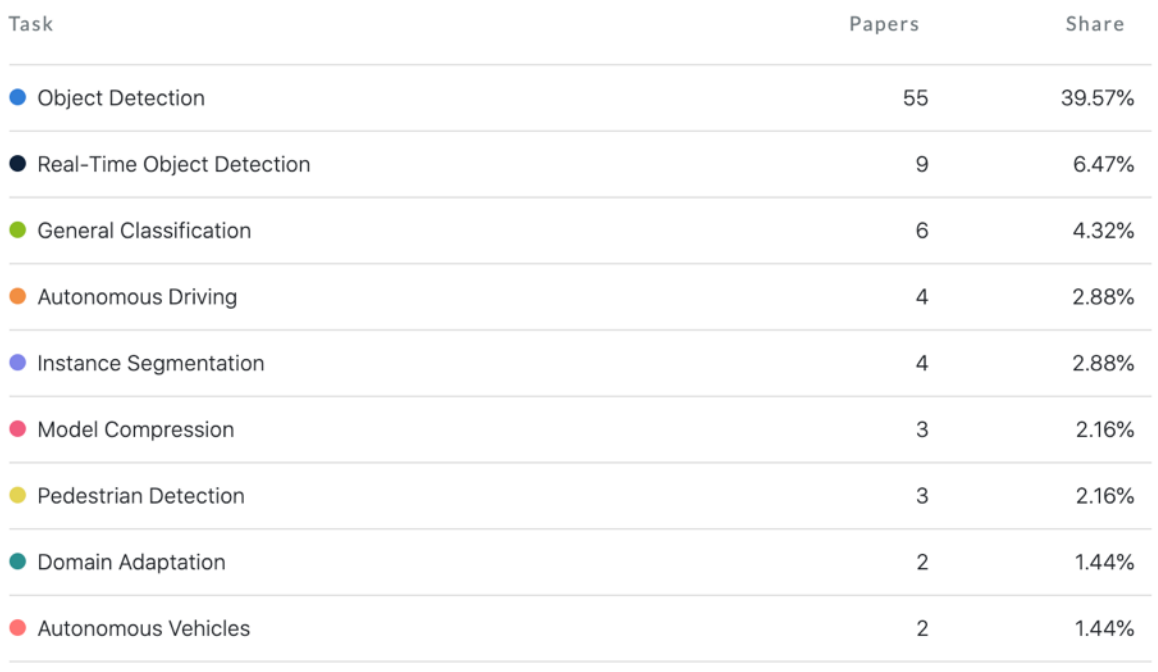
Figure 52. Bounding Boxes and KeyPoints Localization using Detectron2.

The next drawbacks are more complicated to accept but it is important to know they exist in order to understand the algorithm. There are two main difficulties. It is complicated to detect very small objects and it is complicated to detect objects that are very close together. These two problems are derived from the grid approach. Since the image is divided into a grid, if the objects are too small, in each grid there might be some of them and they will not be predicted as independent objects. This is a complication, but it is manageable, since we want to detect humans in public spaces, we assume that the camera and the algorithm will be focused on the nearest people. It is impossible to accurately cover all the depth of a public space and, therefore, the smallest sized objects. The second drawback, the one that makes it difficult to detect objects too close together is a little bit more difficult to assume. Obviously, this problem is logically because of the grid also. Just as before, using a correct grid is key to managing it (It is very difficult to avoid it). Depending on the scenario where the inference is going to be done, the grid size can be fine-tuned to the specific scenario. Moreover, in some cases it can be changed just for the inference but, obviously, accuracy and running time could be affected by it, since the model would be trained with another grid size.

5.3.2 YOLOv3

YOLO is an algorithm that has been present for a considerable time, taking into consideration the youth of this area of research. There is a first and second YOLO version but they are very outdated and I do not think that it is worth it to try those earlier versions too. That is why it is a good reason to get started by trying YOLOv3. The only relevant information from those previous versions is that the backbone used was a 19 layer convolutional network and that the mAP is 48.1, as outlined in both the official website of YOLO[41] and the paper [39].

To put into context all this here are some metrics from the state-of-the-art website for this research[42], that provide a good summary of the use and tasks of YOLOv3.



Task	Papers	Share
● Object Detection	55	39.57%
● Real-Time Object Detection	9	6.47%
● General Classification	6	4.32%
● Autonomous Driving	4	2.88%
● Instance Segmentation	4	2.88%
● Model Compression	3	2.16%
● Pedestrian Detection	3	2.16%
● Domain Adaptation	2	1.44%
● Autonomous Vehicles	2	1.44%

Figure 53. Tasks done with YOLOv3

[41] https://pjreddie.com/darknet/yolo/
[42] Papers With Code

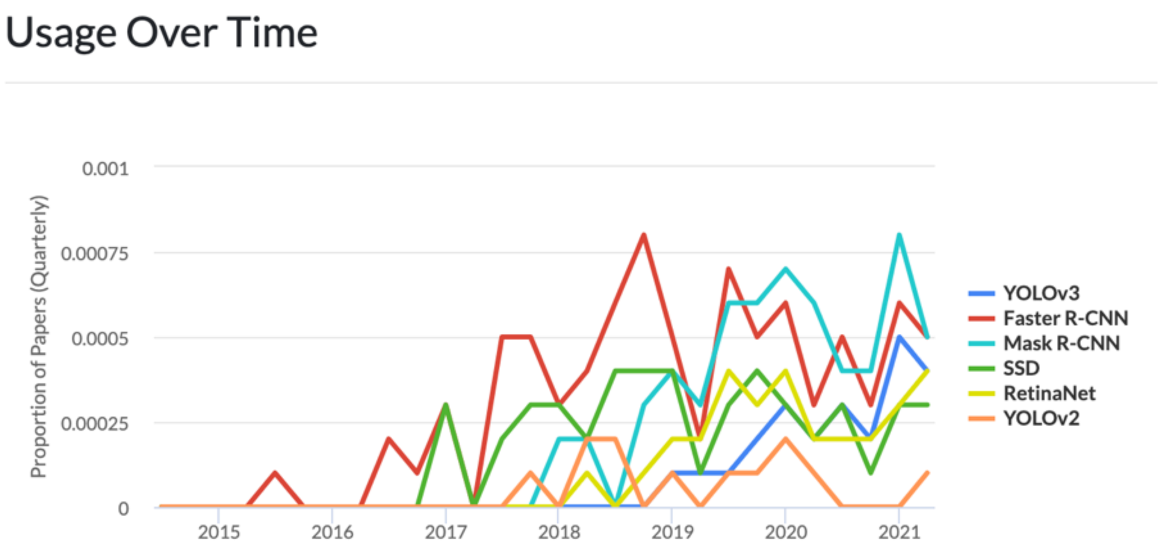


Figure 54. Usage of YOLOv3 over time and Comparison

5.3.2.1 YOLOv3 - Structure

This YOLO version, as explained in the paper [39], is slightly slower than earlier versions, which is very logical since it is a larger model, but it has a higher mAP on all the benchmarks tested such as COCO, VOC etc. [39]. This model includes a multi-scale detector that I will dive deeper later on, a stronger feature extractor as backbone and a slightly different loss function.

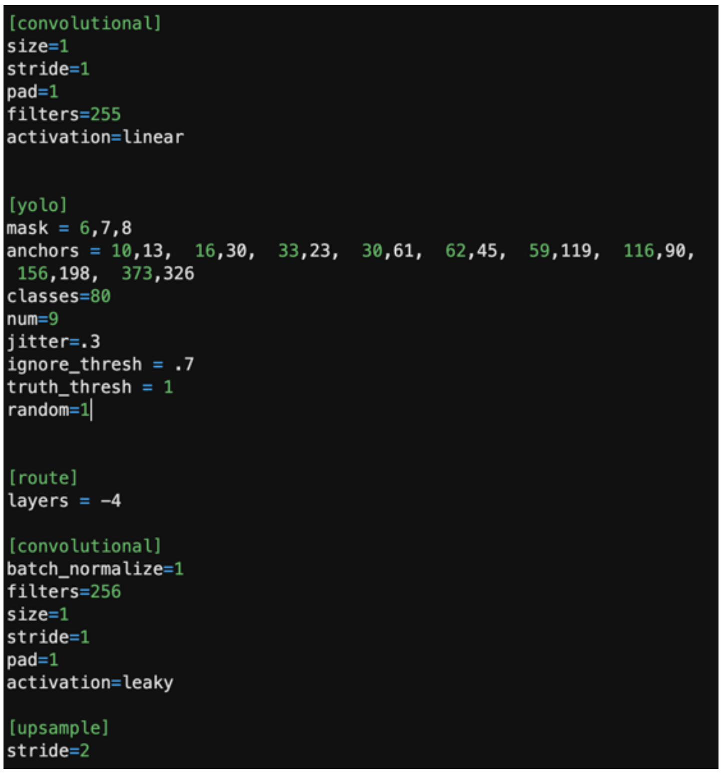
- **The feature extractor**: The previous versions of YOLO used a 19-layer architecture, but as mentioned in this section´s introduction, the algorithm had some struggles detecting small objects because of the down sampling of the input image to fit it in the network and the loss of features between layers. The new network uses what they called Darknet-53. It is a mix of convolutional layers and residual networks that works as follows: There are consecutive 3x3 and 1x1 convolution layers followed by a skip connection (The residual part), which allows to not diminish the gradient too much when propagating information. The combination of the 53 convolutional layers, the residual layers and the detection head result in a total 106-layer fully convolutional architecture.
- **The multi-scale detector**: Derived from the name, the detector needs inputs with different scales. Depending on how the structure is defined, the detector takes the output of 3 different parts of the network and defines feature vectors of different dimensions (13x13, 26x26 and so on and so forth). The larger vectors will be used for the detection of larger objects and the smaller ones for small objects. The different detection heads are placed alongside the model and the preceding convolutional

network needs to have a linear activation function, let's not forget this is a regression-based algorithm. In the following image of the actual configuration file I used for the model, how the detection head ([yolo]) is placed throughout the file. In the model I used there are two more detection heads in the model. The shape of the kernel (what in the file is called 'filters') depends on the number of classes to predicts, so for the different datasets, a different file is needed because the filters of the convolutional layer preceding the detection head needs to depend on the classes provided by the dataset. The formula for calculating it is the following:

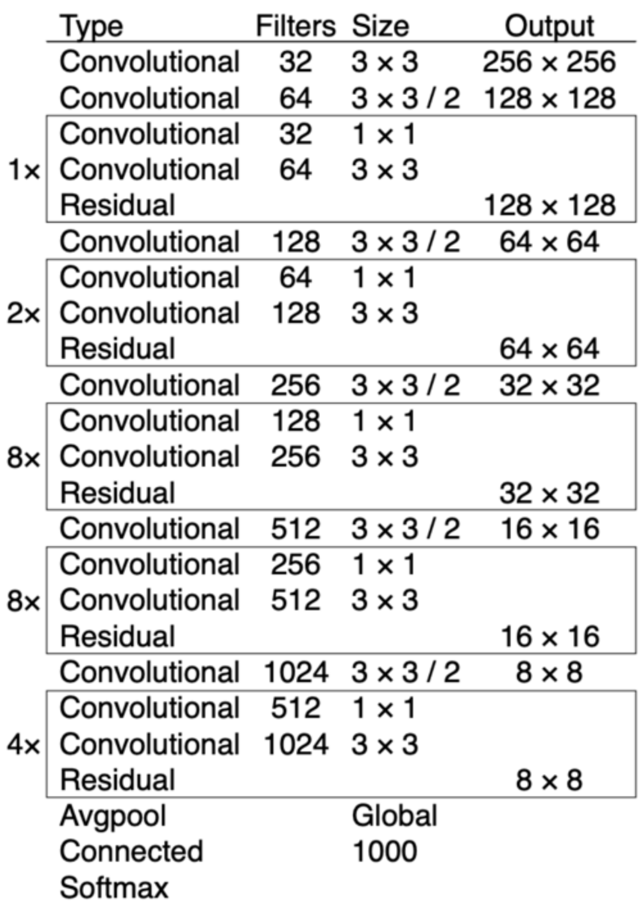
$$filters = \left(B * (5 + numclasses)\right)$$
$$where\ B\ is\ the\ number\ of\ anchors\ (YOLOv3\ uses\ 3)\quad (1)$$

The number 5 on equation (1) refers to the key parameters of the bounding boxes (height, width), (center_x, center_y) and the confidence score (1 per class). For the different datasets this value will change like this: KITTI has 9 classes so the number of filters or kernel size will be 42. Caltech and CrowdHuman have 2 classes so this value will be 21.

In the file, the number of anchors is 9, you can find the explanation here. The values of the anchors are the ones specified in [39].


```
[convolutional]
size=1
stride=1
pad=1
filters=255
activation=linear


[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,
 156,198,  373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1


[route]
layers = -4

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2
```
Figure 55. Detection Head in the Configuration File.

- The loss used in previous versions of YOLO was the sum of squared error. It is very ineffective when doing multi-class classification. That is why in this version the authors decided to use binary cross-entropy to measure loss in the predictions.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 56. YOLOv3 architecture [39]

The final architecture defined in the authors' paper has the detections placed after the 82nd, 94th and the 106th (the final one) layers, respectively. As shown in the image, the images are scaled differently in each detection stage, allowing smaller objects to be detected.

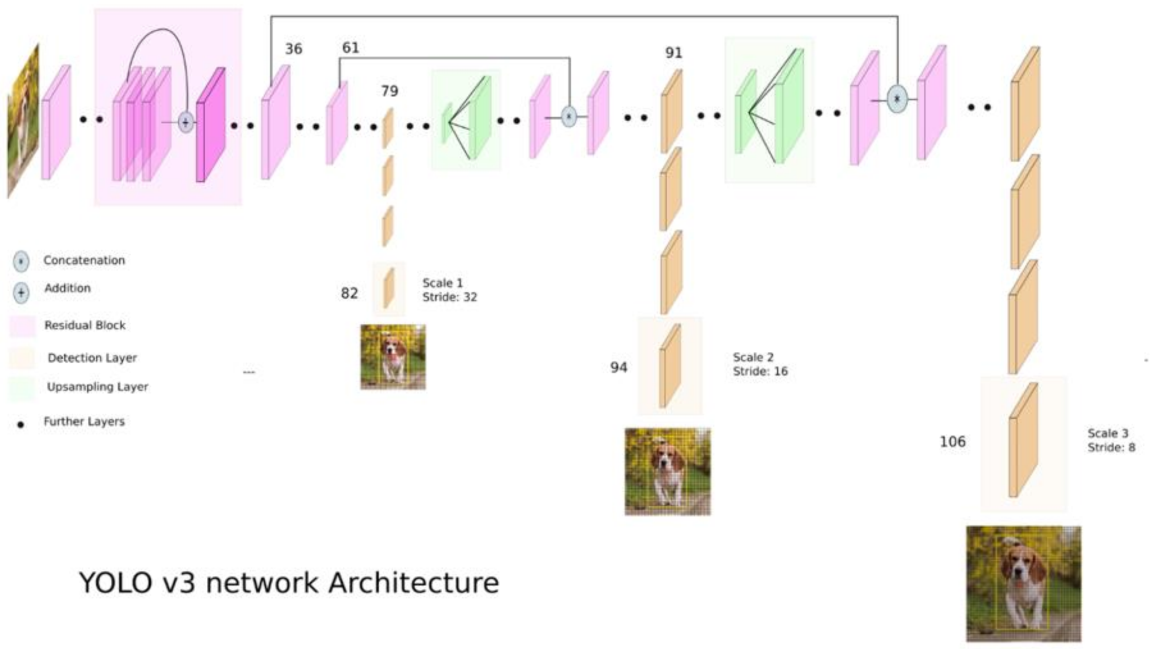


Figure 57. Complete YOLOv3 Architecture[43]

5.3.2.2 YOLOv3 - How It Works

The objective of the network is to predict bounding boxes of each object and the probability of this object belonging to each possible class in the dataset. The input image is divided in a *SxS* grid, and, by the formula explained earlier, there are [B*(5+num_classes)] filters.

As per the paper, the model predicts the class of the object whose center of the bounding box lies inside the grid cell. Therefore, the output of each forward pass through the network is a 3D tensor of dimension [S, S, [B*(5+num_classes)]].

Since the objects to detect are not squares, the authors use a term called “anchor boxes”. They are predefined boxes with an aspect ratio set defined by a K-Means clustering on the entire dataset. YOLOv3 uses 3 anchor boxes per detection scale and the center of these boxes are the same as the centroid of the grid cells. In total, since there are 3 detection scales, YOLOv3 uses 9 anchor boxes.

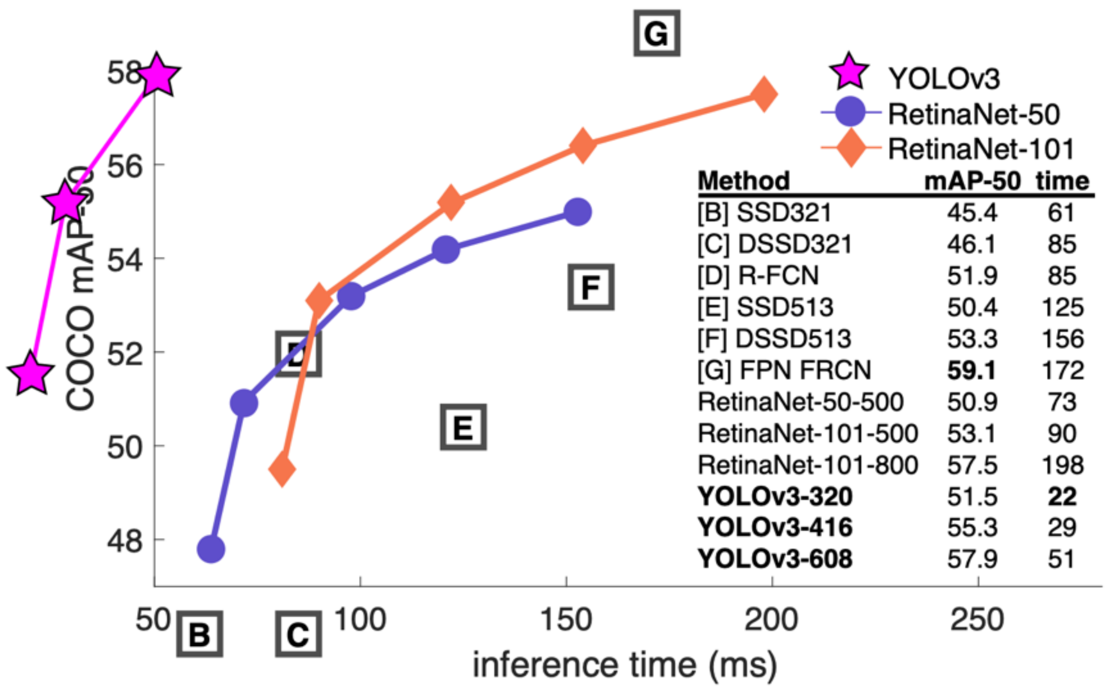
There is a chance that after the single forward pass there are lots of bounding boxes for the same object (same centroids). What is needed is the box better suited for this object. The authors use what is called Non-Max Suppression to select the better bounding box. A threshold can be defined to just clear all the bounding boxes with confidence scores lower

43 https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

than this threshold. After this first stage, it orders in descending order the bounding boxes and determines that the appropriate one is the one with highest confidence score. After eliminating the boxes for this grid cell, it needs to eliminate the rest by doing the IoU, Intersection over Union calculation. IoU is a metric to measure the distance between the ground truth boxes (the labels of the dataset) and the predicted boxes. Its formula is as follows:

$$IoU = Area\ of\ Overlap / Area\ of\ Union \ (2)$$

If the overlap and the union is very similar (IoU approximately 1) it means that the bounding box predicted is very accurate. An IoU of more than 0.5 is considered a good prediction, so I am going to start trying the models with that threshold and a 0.4 confidence threshold.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 58. YOLOv3 Performance Comparison [39]

5.3.2.3 YOLOv3 - Training, Performance and Results

- **COCO Dataset:**

The first combination tried is the YOLOv3 model with the COCO dataset. It is the benchmark dataset for this type of tasks, so I believe it is best to start with it and later on see if any of the rest of the datasets provide a better result.

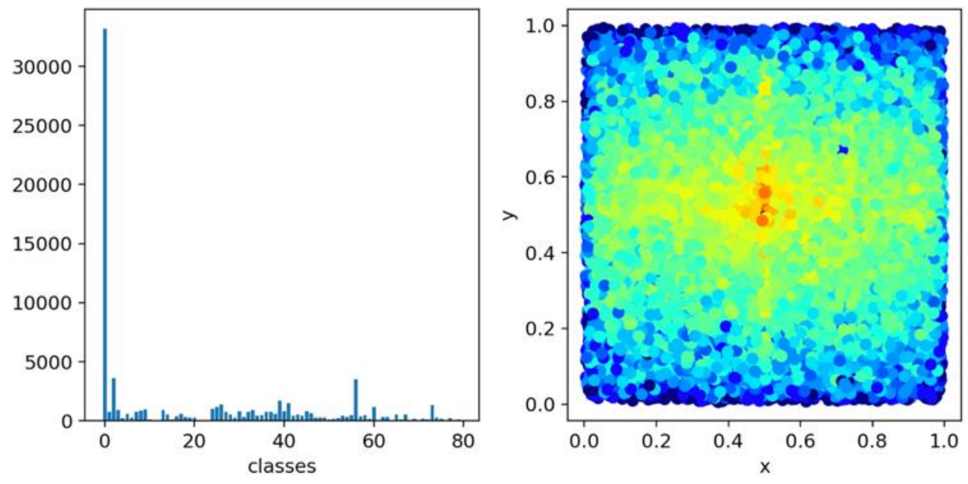


Figure 59. Exploratory Analysis on COCO – Class Distribution & Center Coordinates

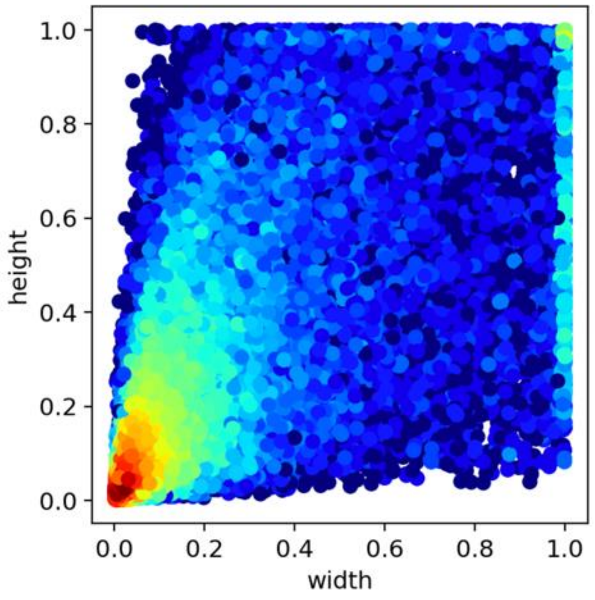


Figure 60. Exploratory Analysis on COCO – Height & Width of Boundin Boxes

I have already mentioned how all the datasets are distributed generally, but above is a visual representation of it. The exploratory analysis shows on the top left the class distribution. As shown, the dataset is very unbalanced because of the filtering I have done so I only keep the images with a ‘person’ label. This is the most probable reason for the low mAP with this dataset, as explained later. However, with the other two representations it is shown that the values for the center coordinates of the bounding boxes are very sparse along the dataset, meaning that the center coordinates of the bounding boxes can appear in lots of different locations depending on the image. Lastly, in the last representation it can be shown the ‘square’ property of the bounding boxes. These properties show that the labels and bounding boxes of the dataset are very different but, most bounding boxes are close to be squares, but in this dataset, you can find everything. The influence of the larger number of ‘person’ labels is clear if you focus on the green ‘diagonal’ on the last image. Those labels are all corresponding to the people tagged in the dataset that have a bounding box that is much more tall than wide.

Training is done for 100 epochs and with a IoU threshold of 0.5 and a confidence threshold of 0.4 as said before. The following image is the mAP at an IoU value of 0.5.

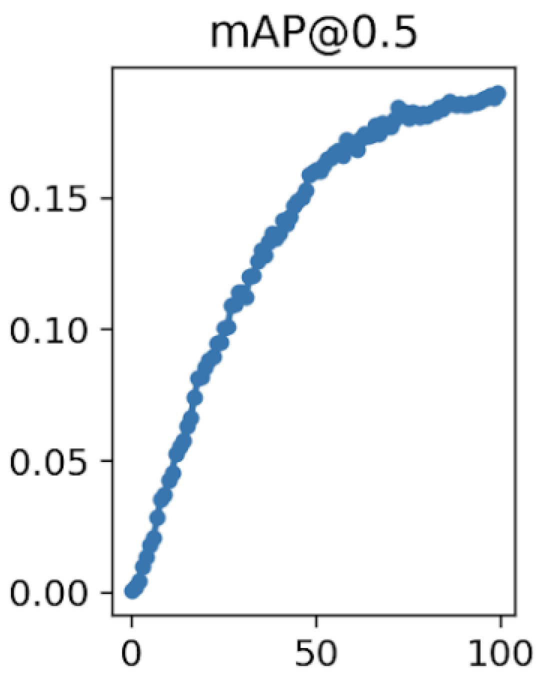


Figure 61. mAP at IoU = 0.5 of YOLOv3 on COCO dataset

The first thing worth noticing is that the model has not stopped growing yet with 100 epochs so it might need a little bit more training. The second aspect worth mentioning is that the mAP value is considerably lower than what the paper promises. I think that there is an explanation for this. Since the dataset is filtered with only the images with people in them, this might have made the dataset completely unbalanced. For example, in the training dataset there might not be any boats images that the model can learn. However, in the validation dataset there might be an image of people on a boat. Since the classes are the same as before the filtering, the model will try to predict the boat class, and, since there are probably not enough boat images in the training set, it is going to fail that prediction. If the predictions of people are correct, missed predictions of other objects are not a problem. A decision wether this model is good enough or not will be made when all results are available.

Since the model is not far away from stabilizing, I think it is better to change other metrics rather than the epochs to train again. I decided to change the IoU threshold and see if the mAP changes. For this next tryout an IoU threshold of 0.65 is used. Taking into consideration that an IoU value of 0.5 translates to good predictions, let's see if having the model try against a higher IoU increases the accuracy.

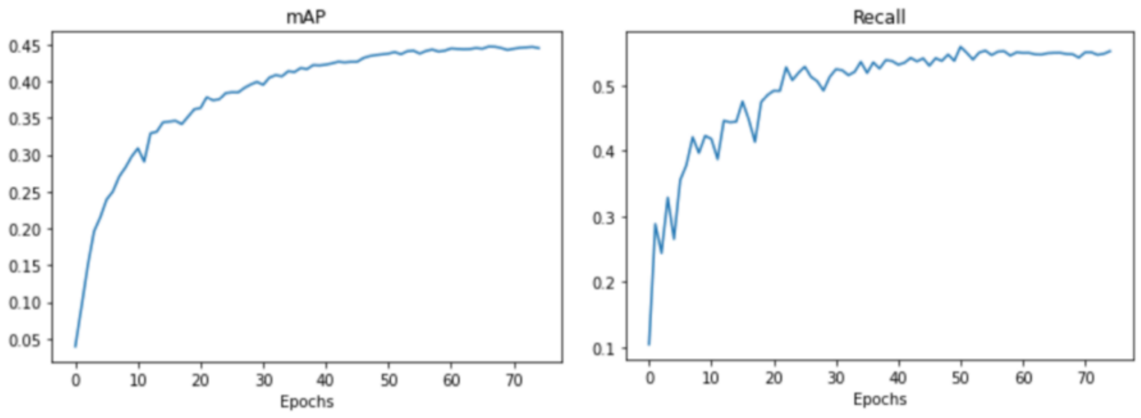


Figure 62. mAP and recall at IoU = 0.65

Looking at the previous result, there is a considerable improvement not only in accuracy but also in model stability since it stabilizes at around 75 epochs. The mAP is exactly 0.4449, which is not that far from the best result that the paper [39] shows, which was 0.571 for images in similar size than mine, which are 640x640. The recall obtained with these model parameters is as shown above. Considering that there is still the inconvenience of classes not being properly trained as explained before, a recall of almost 55.1% is considered a good result.

- **Caltech Dataset:**

Now the same model is tried on the Caltech datasets. As mentioned earlier, some parameters on the configuration file of the model need to be changed. The number of filters (kernel size) is now 21 because this dataset only contains 2 classes, person and people.

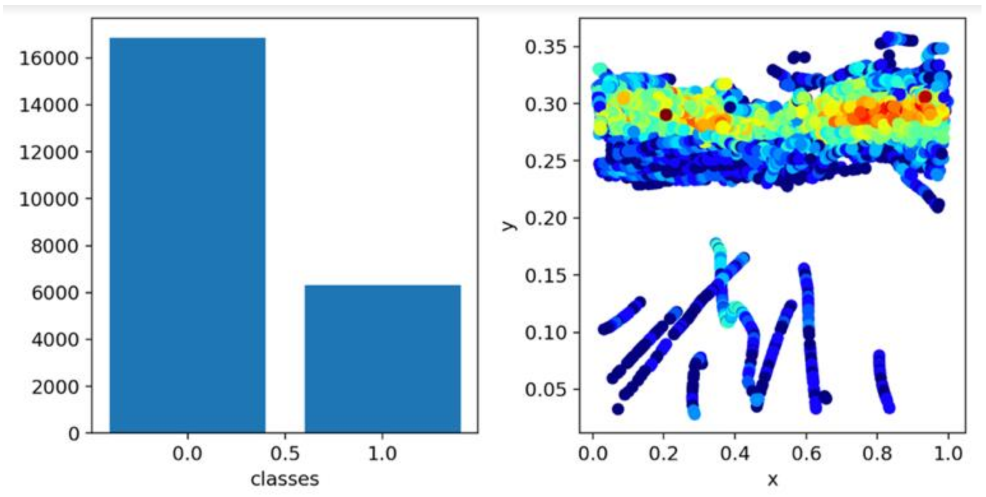


Figure 63. Exploratory Analysis on Caltech – Class Distribution & Center Coordinates

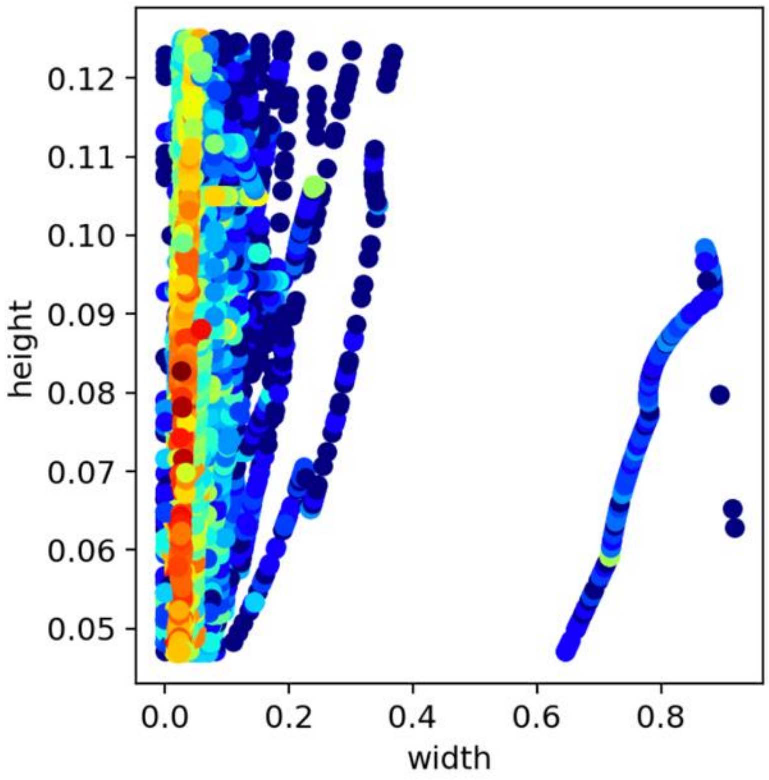


Figure 64. Exploratory Analysis on Caltech – Height & Width of Bounding Boxes

The exploratory analysis on the Caltech dataset shows clearly why it struggles considerably with YOLO. The two classes are ‘person’ and ‘people’, which is a clear warning that it is not going to be able to differentiate people close together. On the second visualization it is shown that the center coordinates of the bounding boxes in the image are almost every time in the same place of the image, and since they are very rectangular, it does not help YOLO at all. The third image shows the shape of bounding boxes, being all of them people it is logical that all of them are tall and narrow except the ‘people’ labels, that are separated from the rest.

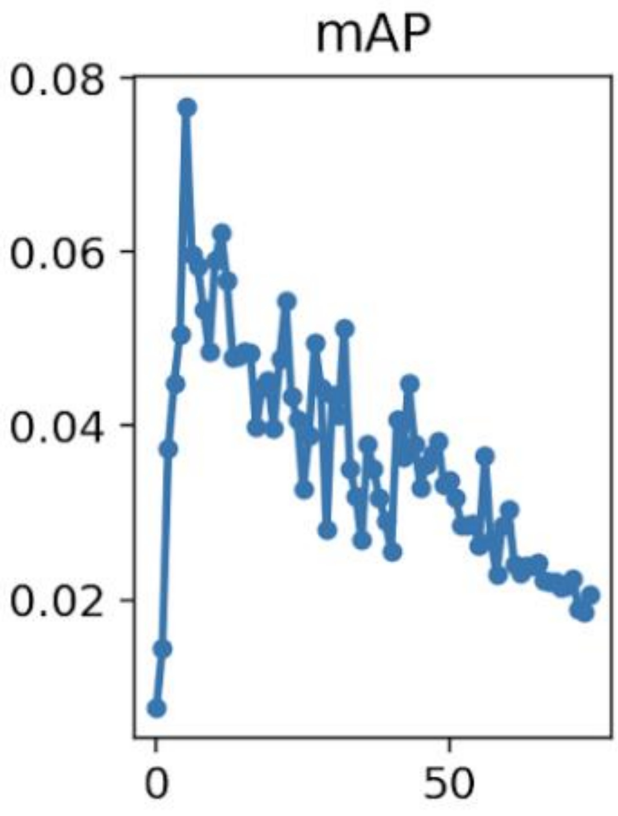


Figure 65. Result of Caltech Dataset

After several tries changing the different parameters and even evolving the hyperparameters, I have been unable to find any good result with this dataset. I assumed beforehand that the results would not be as good with this dataset mainly because of the reasons I explained earlier. First one is the poor quality of the images in the dataset, second of all, images are very far away so the algorithm really struggles when trying to extract features there. Finally, the format is far from optimal for the model even after squarifying the images. There is something I am not identifying because I have seen good results on this dataset in some publications like this, but I guess I am missing. It is true that most models using this dataset are RCNN based.

- **KITTI Dataset:**

The next dataset would be KITTI. This dataset, at least in training, should improve the results from previous models. That is because all the images are very similar within each other, there are not many class labels (just 9) and, even though the images are not totally squared, they are not as bad as the Caltech ones as you can see in the comparison below. Adding to that, the resolution in KITTI is much higher than in Caltech, a fact that can help very much even though they are rectangular images.

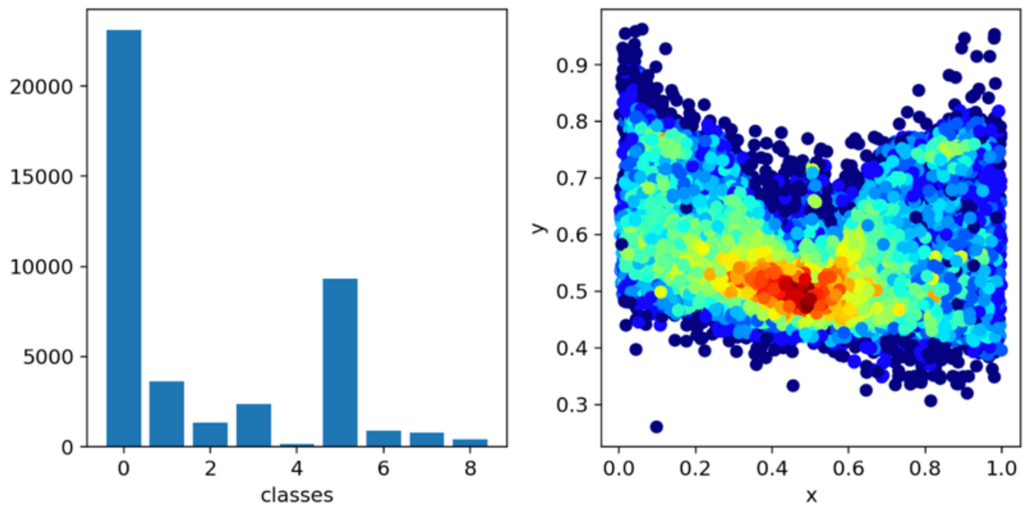


Figure 66. Exploratory Analysis on KITTI – Class Distribution & Center Coordinates

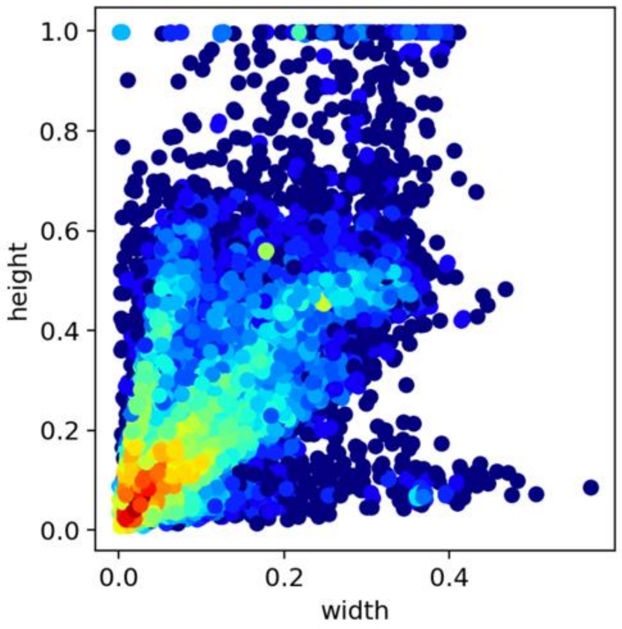


Figure 67. Exploratory Analysis on KITTI – Height & Width of Anchor Boxes

As mentioned before, KITTI is similar to Caltech, but much better in terms of having close to 'squared' images. In the second image the distribution of center coordinates along the images is way richer, allowing the model to struggle less with the input images. The size of the bounding boxes is also very variable, and that is because this dataset contains labels for 'cars', 'bicycles', 'trucks' etc. The class distribution shows one of the reasons this dataset may not generalize properly to our problem. Label 1 corresponds to 'cars' and label 5 is 'person'. The model will learn to classify cars very accurately because for an autonomous vehicle, the perspective of this dataset is the only one that is possible. However, that is not the case for people, that can appear in multiple different perspectives.

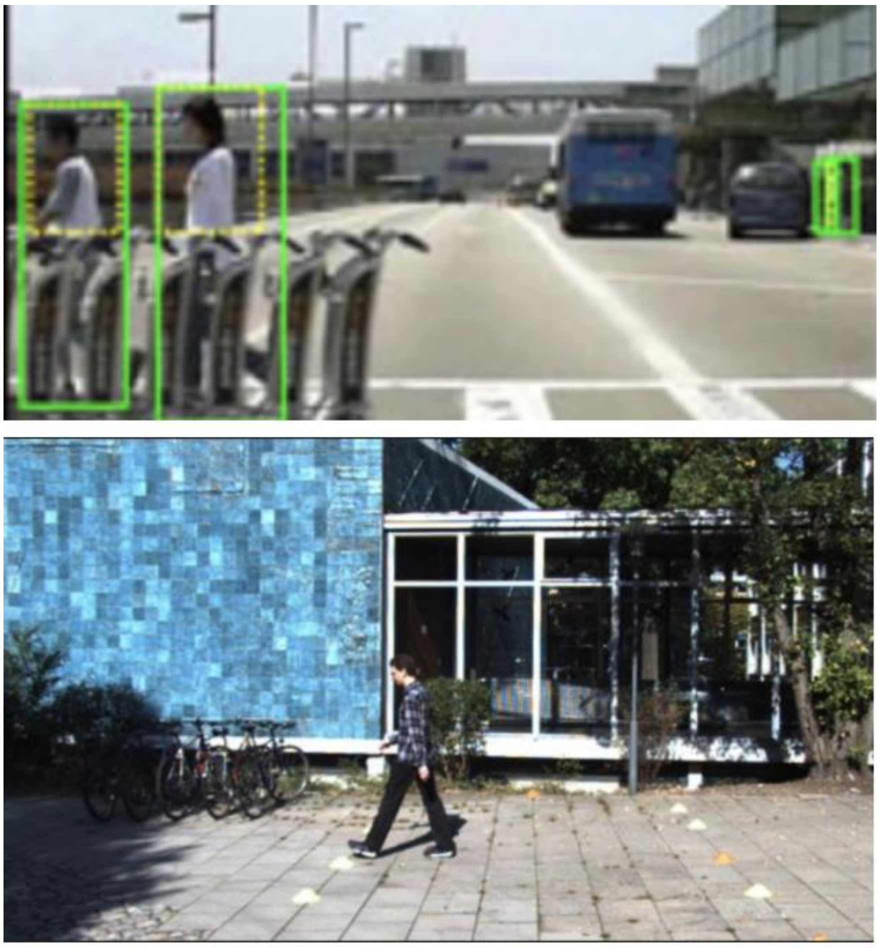


Figure 68. Caltech (Top) and KITTI (Below) examples

The results provided by KITTI using the same parameters as the best model of COCO are the following:

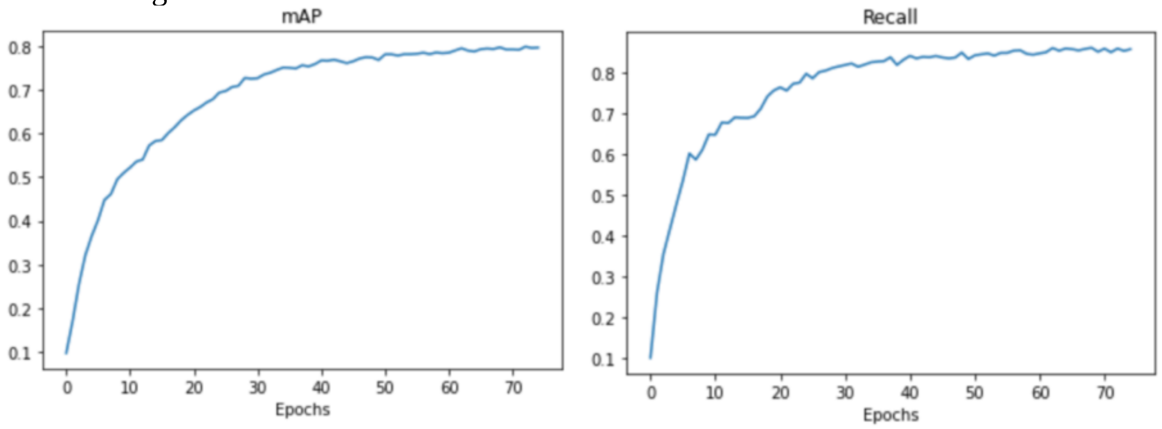


Figure 69. **mAP and Recall on KITTI**

In paper, great results, but, since this dataset is a very specific dataset for autonomous vehicles, all the images in it are from the same perspective and they are focused more on the cars and surrounding aspects of the camera rather than the pedestrians. This characteristic of the dataset may produce generalization issues that will be checked later on inference.

The mAP score is 0.84 which is very very good and so is the 0.86 recall value.

- **CrowdHuman Dataset:**

The last dataset that is going to be put through the training process of this model is the CrowdHuman dataset. Actually, a priori, it is the dataset more suitable for the task that is being treated, so, if the training results are good, this model should be the one used for the rest of the project.

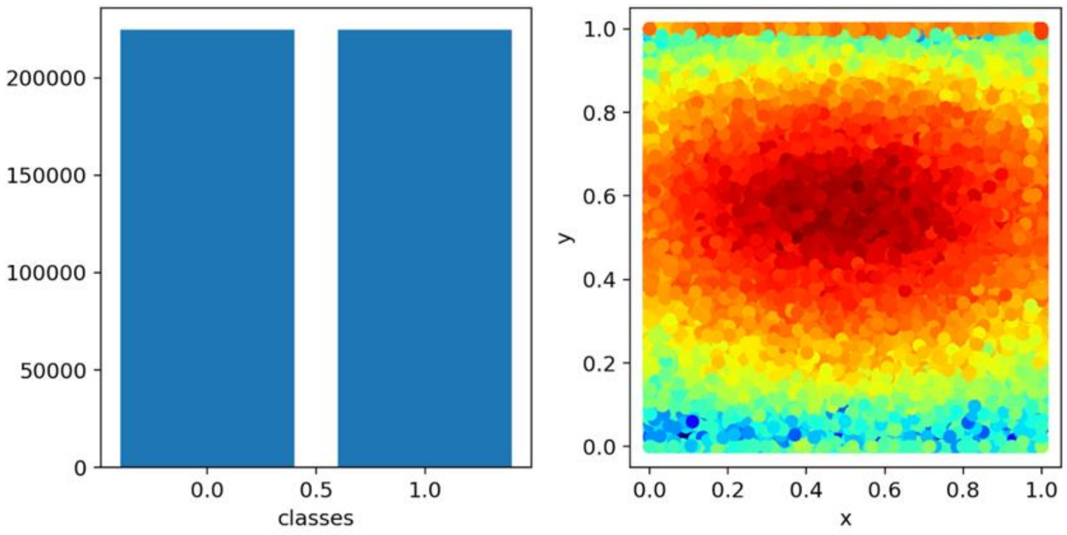


Figure 70. Exploratory Analysis on CrowdHuman – Class Distribution & Center Coordinates

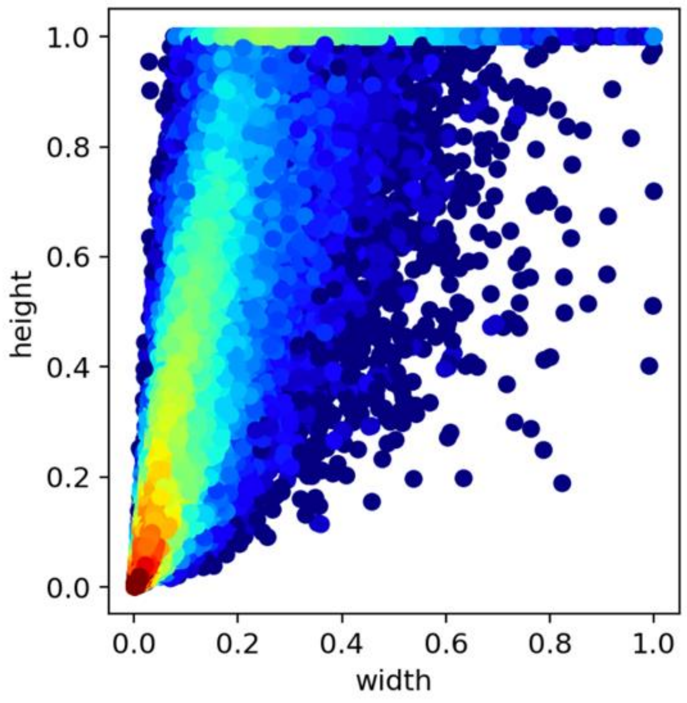


Figure 71. Exploratory Analysis on CrowdHuman – Height & Width of Anchor Boxes

As seen in the exploratory analysis, obviously there are the same number of heads as people, and there are more than 200.000 instances, much more than in all the previous datasets. Obviously, the bounding boxes are way higher than wider and with lots of very small boxes (The heads). The bounding boxes are more or less spread more homogenously than in the previous datasets through the image frame.

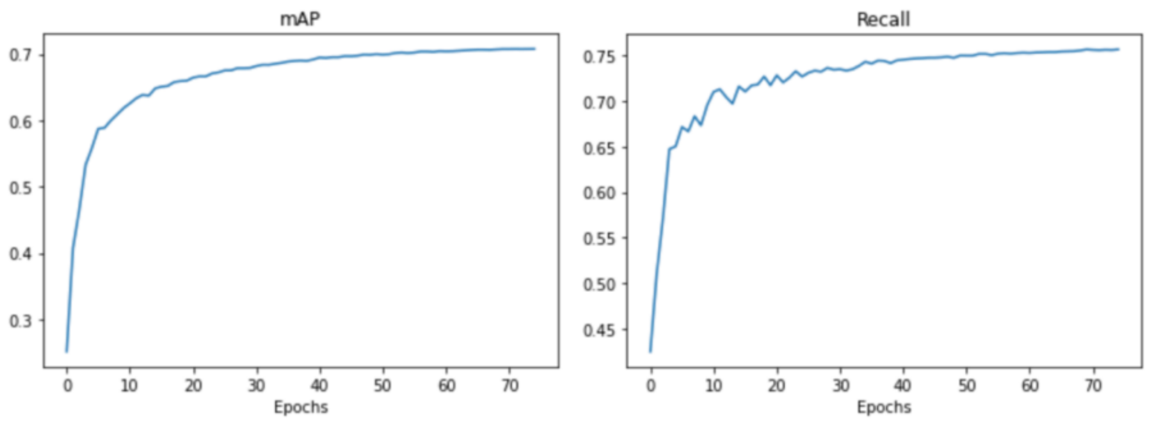


Figure 72. mAP and recall on CrowdHuman

The above results are very good, the best ones so far. Of course, they are not as good as the KITTI results, but that is fairly obvious because of the nature of these datasets. While KITTI is a dataset that is not very variable, being all the images similar between them, CrowdHuman is a much more variable dataset, providing images in very different contexts that can be very helpful for the different perspectives a camera can be placed.

Having a 0.73 mAP and a 0.79 Recall I consider a very good result taking into consideration that the YOLOv3 paper provides only a 0.57 mAP on COCO. Of course, COCO has 80 classes to predict that makes it more difficult but, the point is that a 0.73 mAP for this model is very good.

5.3.2.4 YOLOv3 - Inference

All the links to the full inference results including videos are on Annex II: Inference Results

- **COCO Dataset:**

Here is a batch of images and links to videos where I have tried the model trained on COCO with.

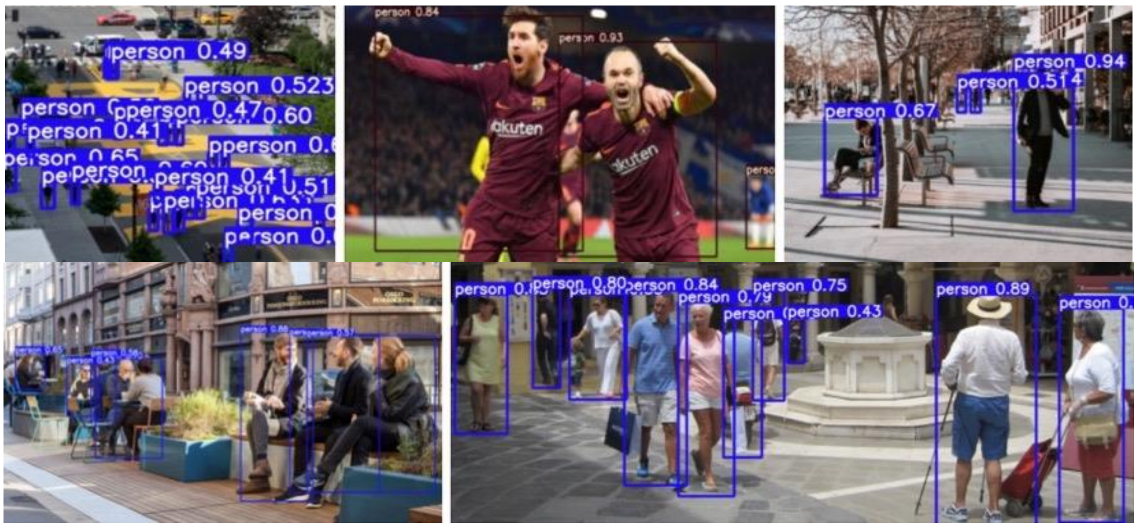


Figure 73. Mosaic of Inference on Images – COCO YOLOv3

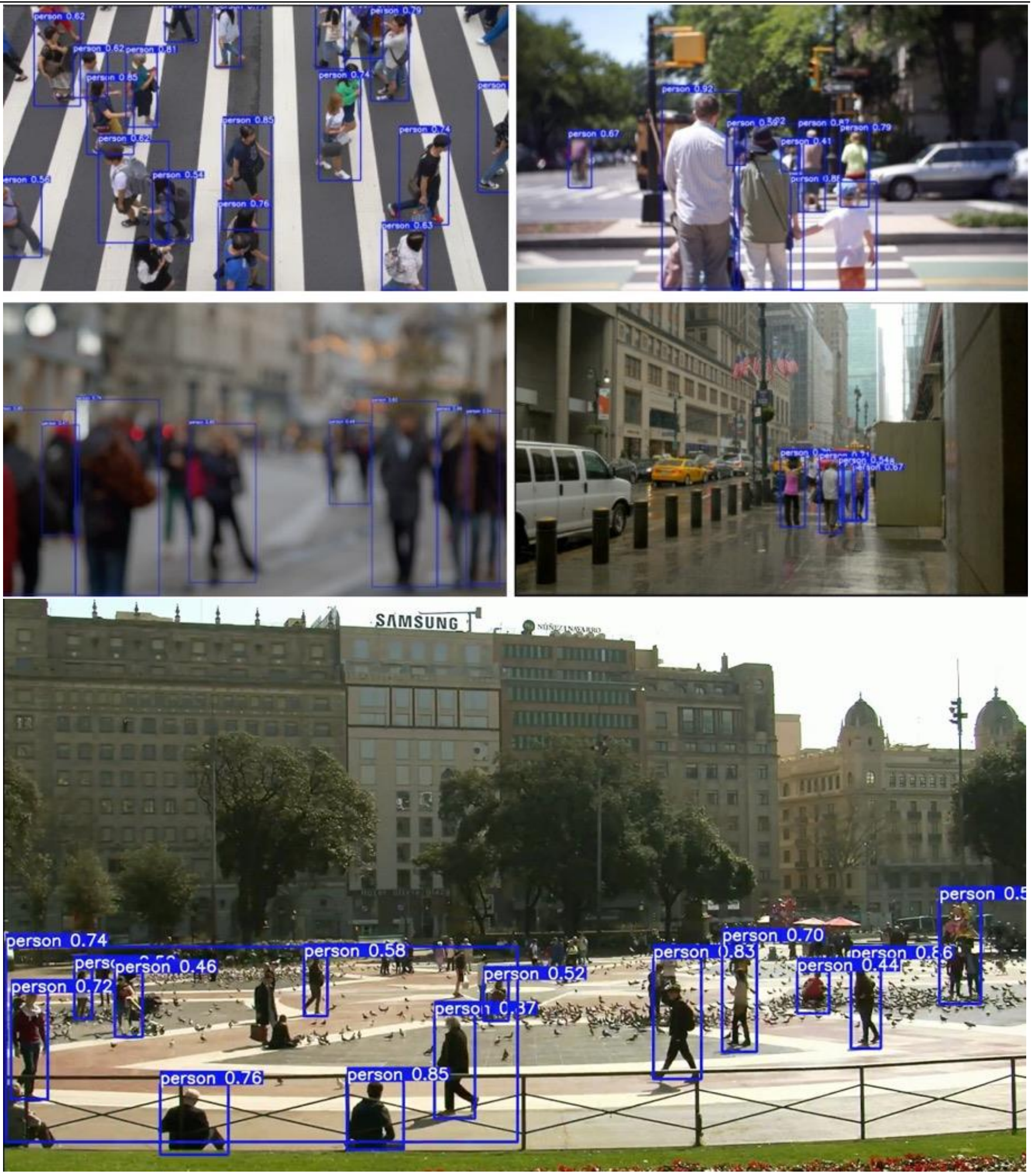


Figure 74. ***Mosaic of Inference on Videos*** ***– COCO YOLOv3***

As expected, this algorithm has some problems, but overall, I think it provides a good result when dealing with open space locations, as shown specifically on the videos. The two main problems it encounters is when dealing with a long-range action, like in the video with the last capture and the first image. It also struggles a little bit with occlusions, especially when the camera is placed perpendicularly to the people that the model tries to identify, but it is not a disaster.

- **Caltech Dataset:**

There is no need to provide results for Caltech because it is not worth it looking at the results from training. The model is not able to detect individuals with the different perspectives used.

- **KITTI Dataset:**

As feared, the model trained in this dataset suffers from a little bit of a generalization problem, as seen in the examples below. It is not a disaster, but it struggles a lot more than the COCO model when dealing with occlusions and different perspectives. It also struggles considerably when the camera is in an environment different from the street that is why in images where the context is not a street the algorithm does not provide a very good result. On the bright side, it has a better result when dealing with people in larger distances, since that type of images are more common on this dataset than in COCO, however, in extremely close range it has some problems. This is very clear in the video corresponding to the last capture and the first image, where you can also see very clearly how bad it detects a person that is not in a walking position since there are multiple people seated on the floor and it does not detect them. In the third image, it is clear that it cannot detect the closest person on the frame. It also struggles with blurry images considerably as seen in the videos. In spite of all these problems, the model could be useful in specific situations where the perspective and distance are aligned to the specifications of the dataset, but I would not use it in a general perspective.

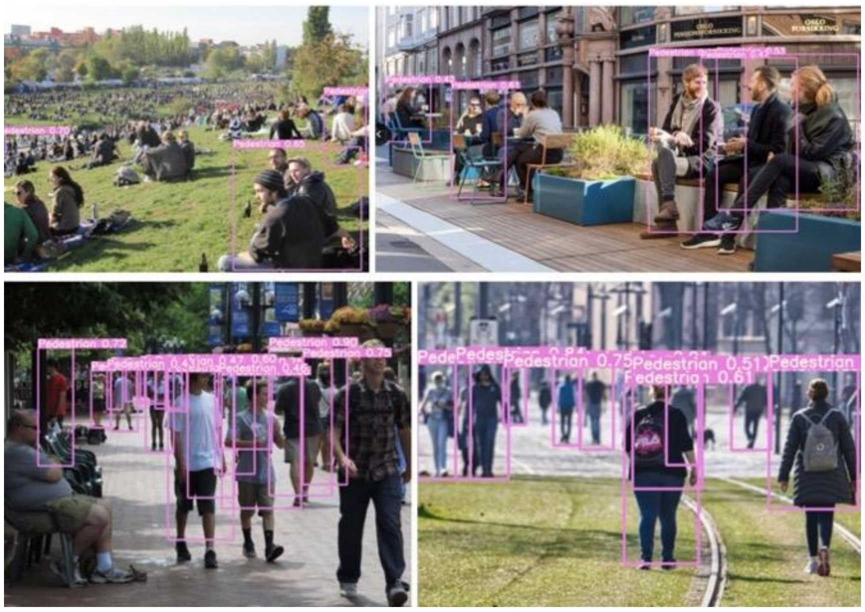


Figure 75. Mosaic of Inference in Images – KITTI YOLOv3

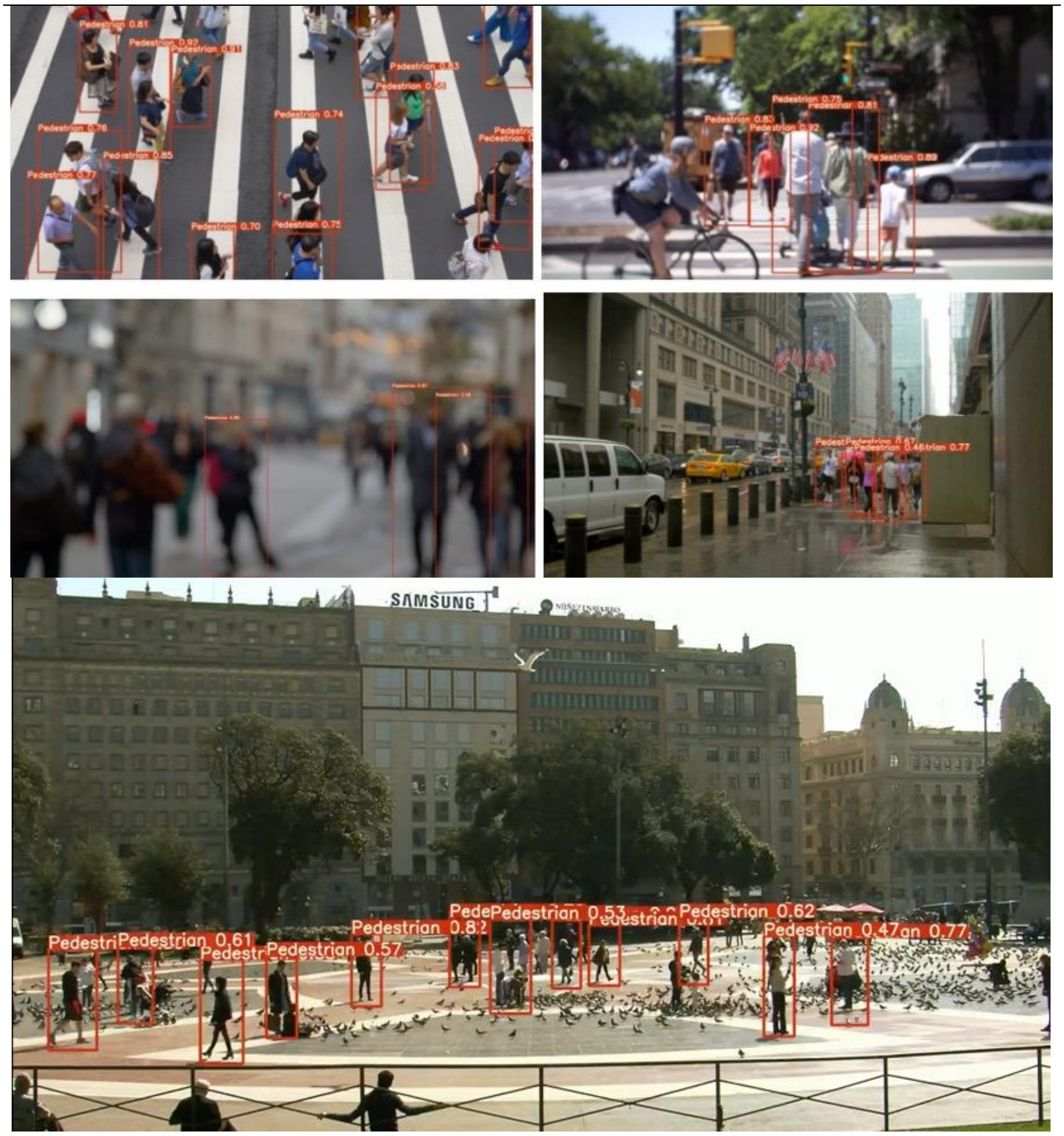


Figure 76. Mosaic of Inference in Videos – KITTI YOLOv3

- **CrowdHuman Dataset:**

Inference with this model and dataset should be very similar to COCO, let's see if there is any notable difference.

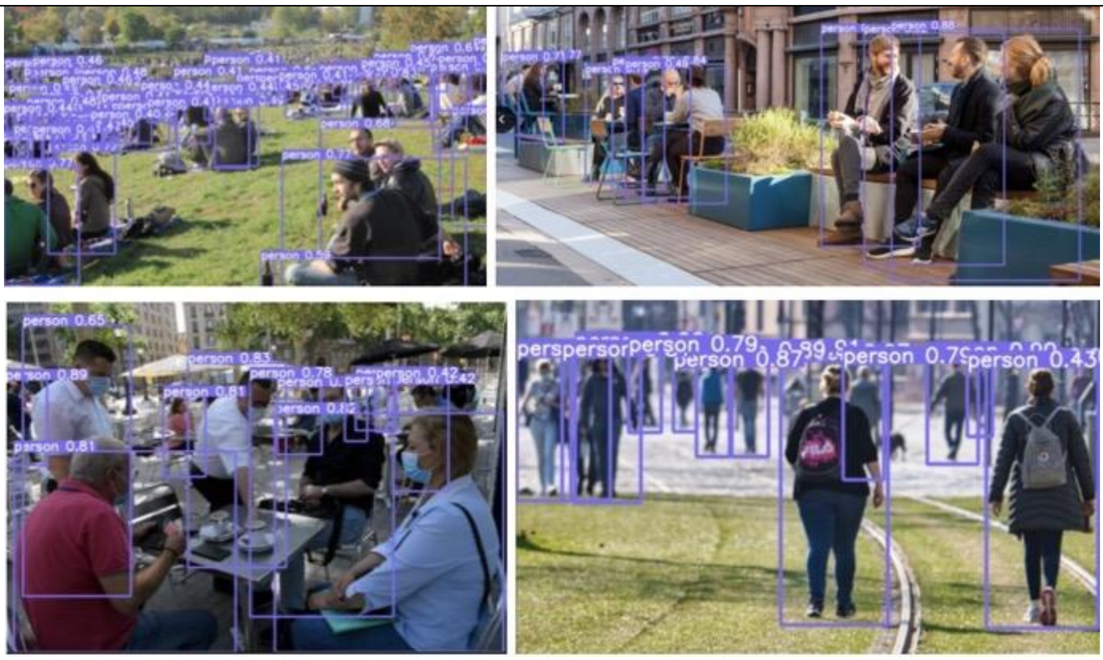


Figure 77. Mosaic of Inference in Images – CrowdHuman YOLOv3

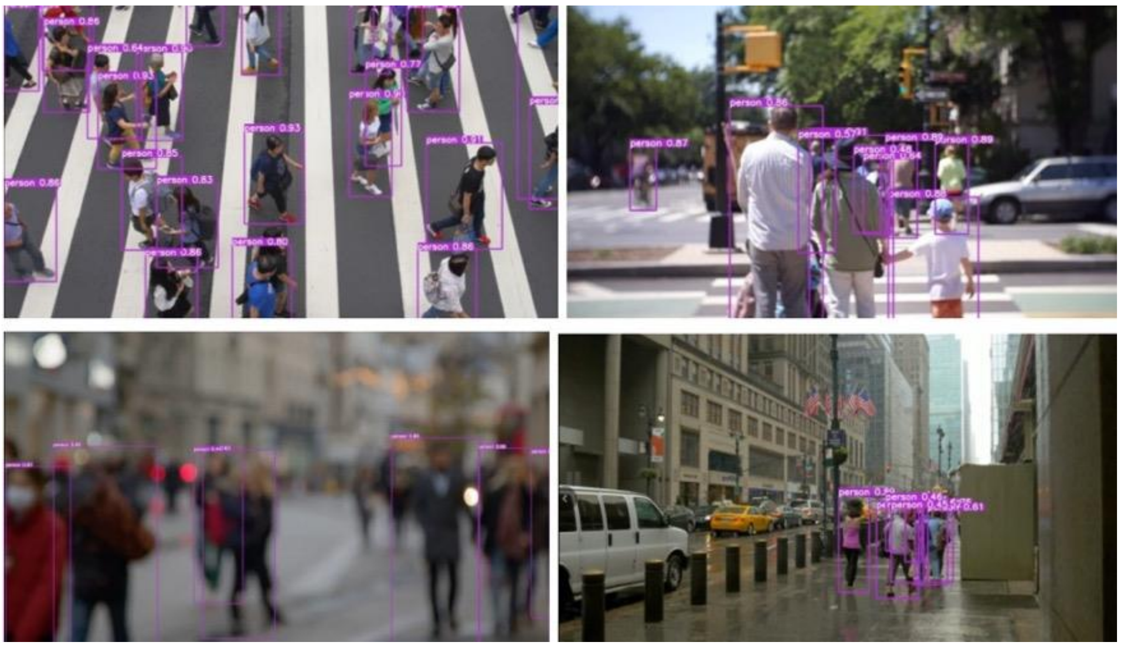


Figure 78. [Mosaic of Inference in Videos](#) – CrowdHuman YOLOv3

As expected, the inference is very similar to COCO. I find it a little bit better when dealing with occlusions and cenital perspectives like the first video. Taking this into consideration and also the fact that this is, according to the previous metrics, a more robust model, I believe that, for YOLOv3 the better model is the one trained on the CrowdHuman dataset.

Just by looking at the images it is very clear that this model combines the strengths of both previous models. It can deal with different positions and perspectives like COCO and fix that problem in KITTI (Just compare the first two images on the KITTI and CrowdHuman mosaics). It also fixes the long-range problem and the perpendicular camera problem with COCO; the improvement is clearly visible in the first and fourth photos in the CrowdHuman mosaic.

It is also able to detect many more people than the other models using the same amount of time (average inference time of 26ms).

5.3.3 YOLOv4

Since YOLOv4 is just an evolution of YOLOv3, I am not going to explain again how it works and the architecture of it, I am just going to explain the main changes between the models and jump straight to the results. All the explanations and metrics are taken from the official paper [43].

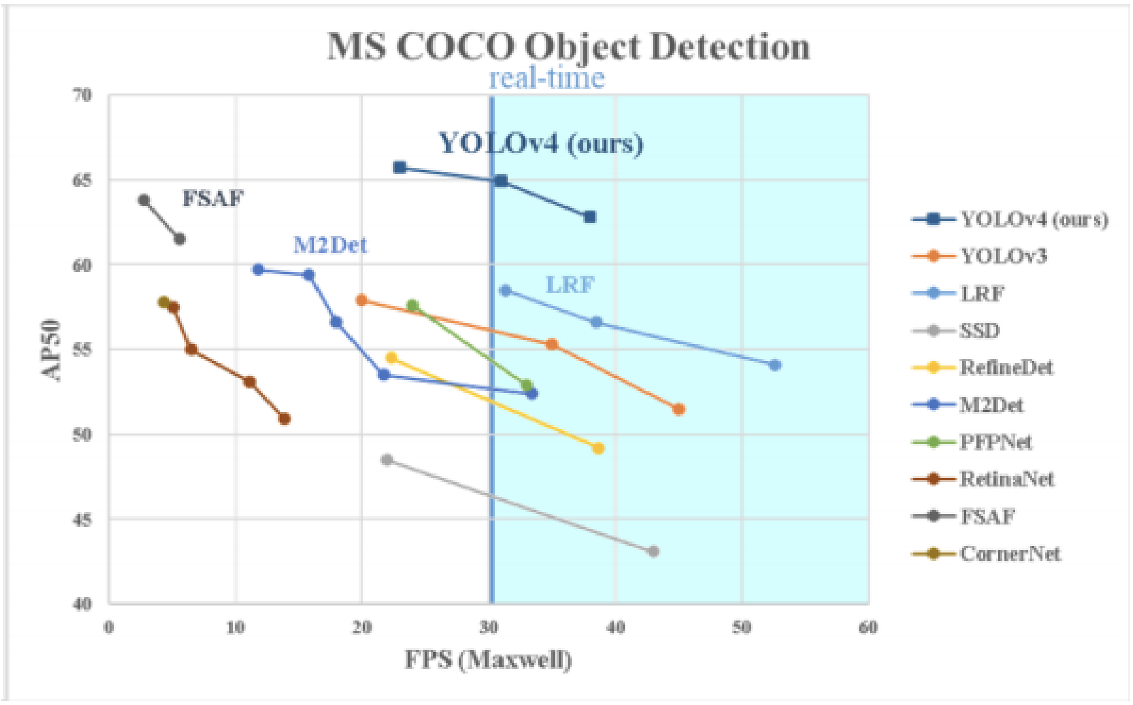


Figure 79. YOLOv4 on COCO [43]

Above is the comparison of performance that the authors provide. This model is supposed to improve the accuracy considerably at a very little cost in time. It uses a similar backbone as YOLOv3 followed by a neck composed of some layers to connect feature maps from different stages, just as the previous version does but instead of using them as input directly into the detector, it processes them with some additional layers. One key difference is that this method also changes the loss method again. From binary cross-entropy it uses a version of IoU loss called GIoU loss, both based on the previously explained computation of IoU. The main difference between GIoU and IoU is that GIoU takes into consideration orientation of the object in addition to the overlap area.

It uses a slightly different backbone called CSPDarknet53. It is very similar to the original Darknet53, the only difference is that it uses a partition strategy to separate the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. This approach allows a larger gradient flow through the network. The structure of a CSPNet is as follows [44]. This architecture is more light-weighted and enhances the variability of the learnt features in the different layers [44].

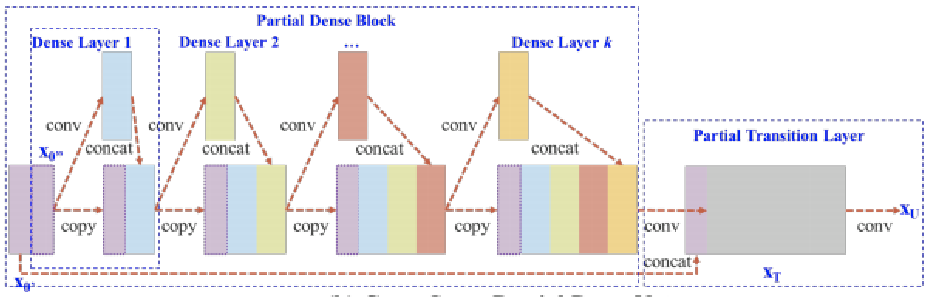


Figure 80. CSPNet Architecture[44]

The other main difference is the neck. The method used for parameter aggregation is PANet to sum the features and parameters from different levels. PANet is chosen because of its ability to preserve spatial information accurately, helping in proper localization of pixels for mask formation. This is done by doing adaptive feature pooling and then concatenating the neighboring layers, making the architecture much deeper. The architecture is as follows:

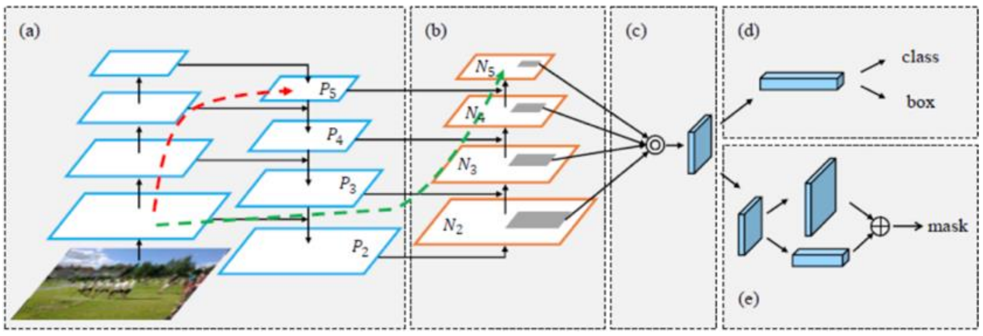


Figure 81. PANet Network[44]

[44] medium.com/clique-org/panet-path-aggregation-network-in-yolov4-b1a6dd09d158

It uses the same detector head as YOLOv3 and the same residual approach, as well as the activations (LReLU, PReLU…), batch normalization and regularization method (Dropout).

These are the main improvements introduced to YOLOv4, let's see if they translate to this task and if it improves the results of YOLOv3.

5.3.3.1 *YOLOv4 - Training, Performance and Results*

For all these new results, the same data exploration conclusions drawn in section 5.3.2.3 apply here.

- **COCO Dataset:**

Again, the same datasets were tried except the Caltech one through YOLOv4, in hope to see some improvement in accuracy.

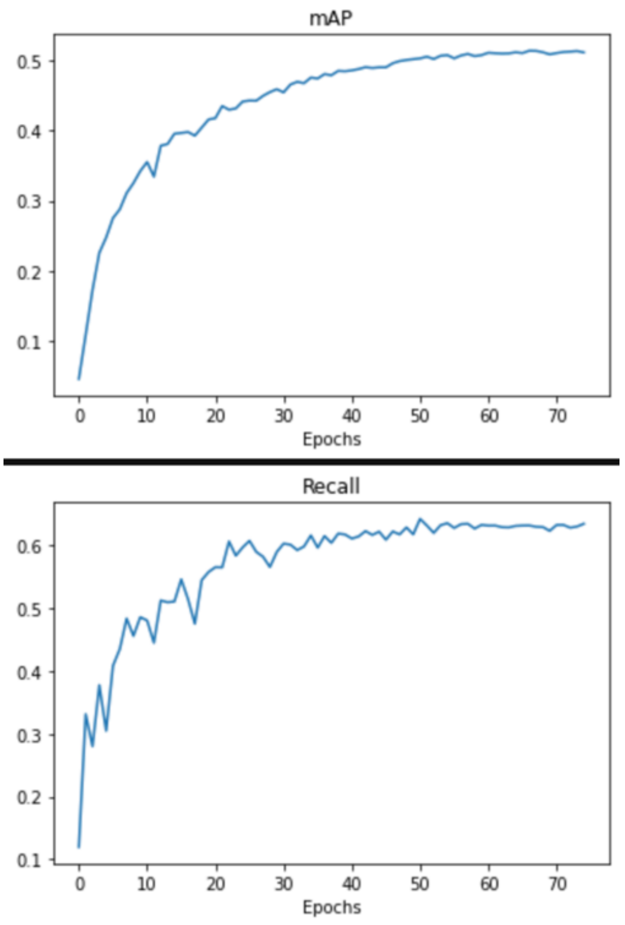


Figure 82. mAP and Recall for YOLOv4 on COCO

Apparently, there is some improvement. Remembering the results that YOLOv3 provided, mAP was 0.4449. Having a mAP with YOLOv4 of 0.51 is a significant improvement. It is true that it stays far from the mAP recorded by the authors of the paper, which was 0.65 more or less, but it is an improvement from the previous method that is all we look for.

Apparently, this model is better, later in inference it is shown if it is visually better also.

- **KITTI Dataset:**

As seen in the result curves below, KITTI has a mAP and recall values very similar to YOLOv3, it does not provide a notable improvement as in COCO. As a side note, the model is more unstable than the one trained in COCO even though the accuracy reached is higher than the COCO model.

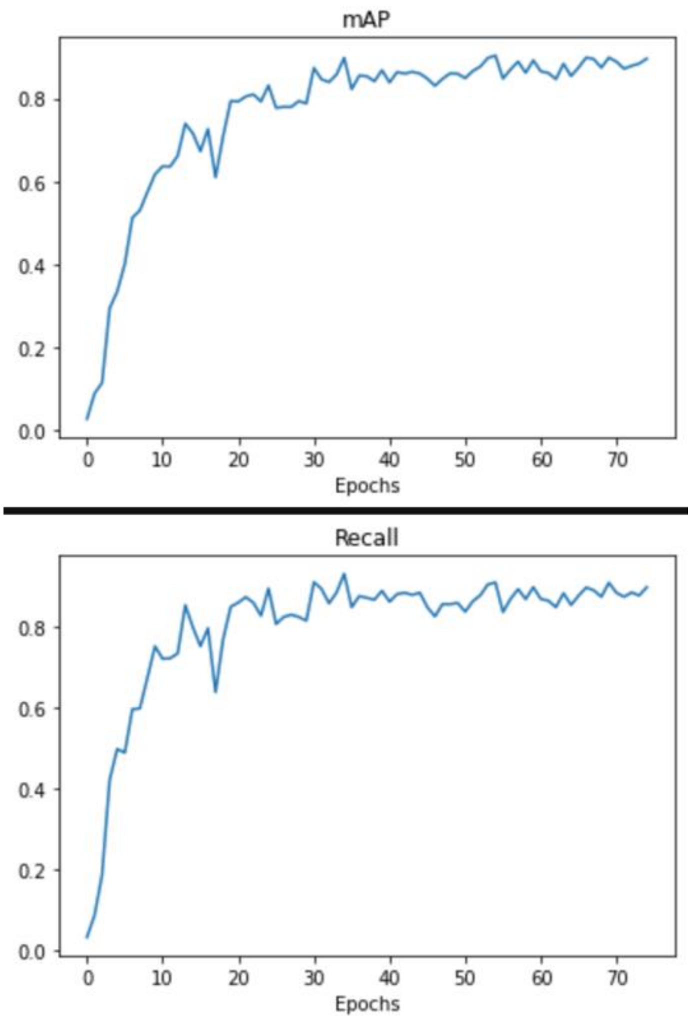
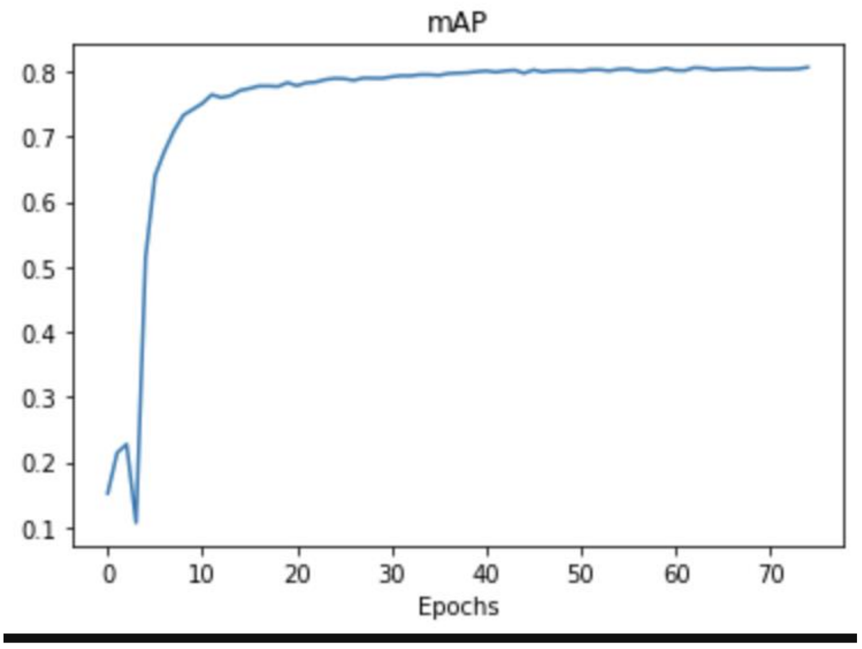


Figure 83. mAP and Recall for YOLOv4 on KITTI

Even though the results of KITTI are better than COCO again, it is expected that it fails in the same way as the YOLOv3 version because it still cannot generalize as well as the models trained in the other datasets.

- **CrowdHuman Dataset:**

Finally, the dataset that provided the best overall results in the previous version of YOLO was tried.



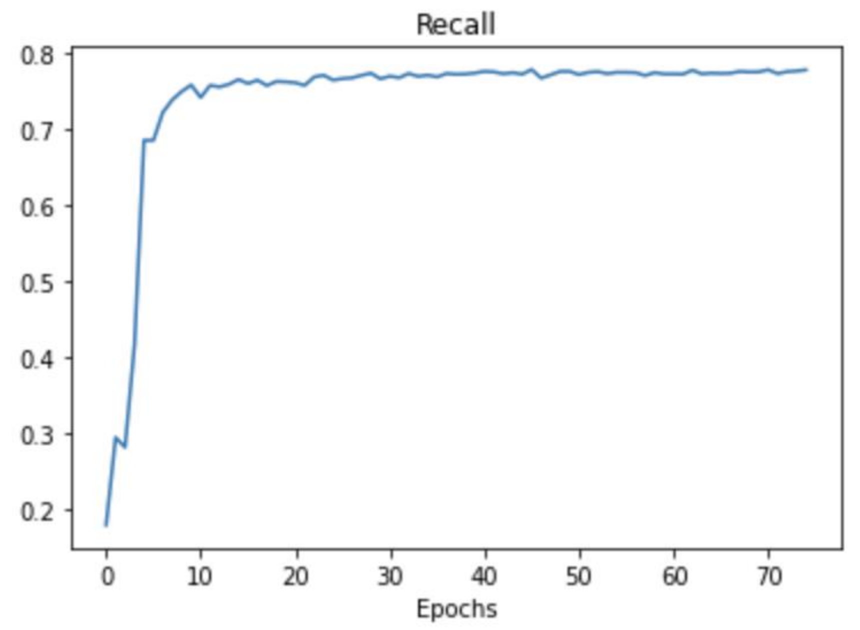


Figure 84. mAP and Recall for YOLOv4 con CrowdHuman dataset

As expected, CrowdHuman again provides very solid results and a very robust model.

5.3.3.2 YOLOv4 – Inference

- **COCO Dataset:**

Since the performance of the models is not much better, I expect that the results are quite similar.

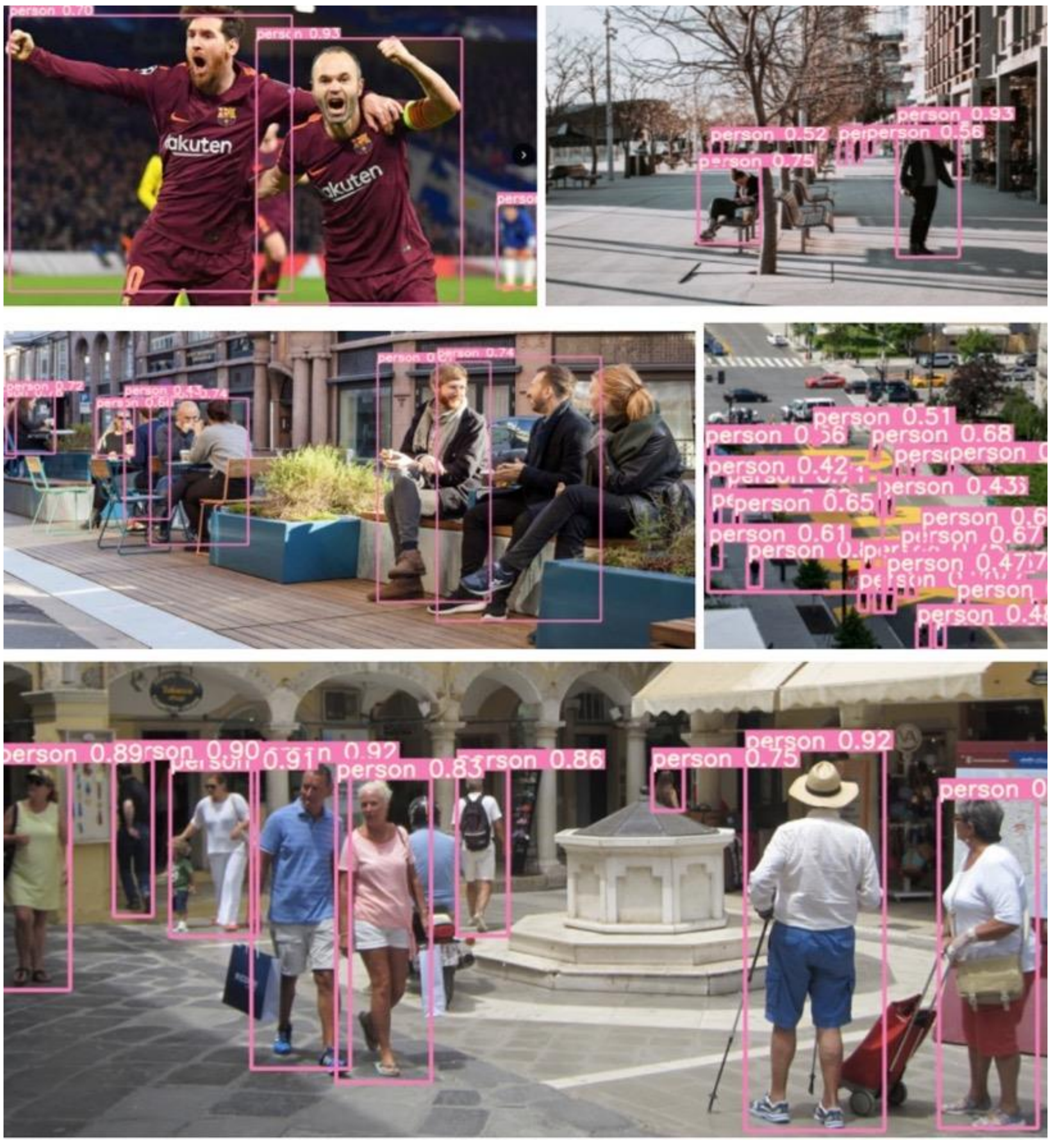


Figure 85. Image Mosaic of COCO on YOLOv4

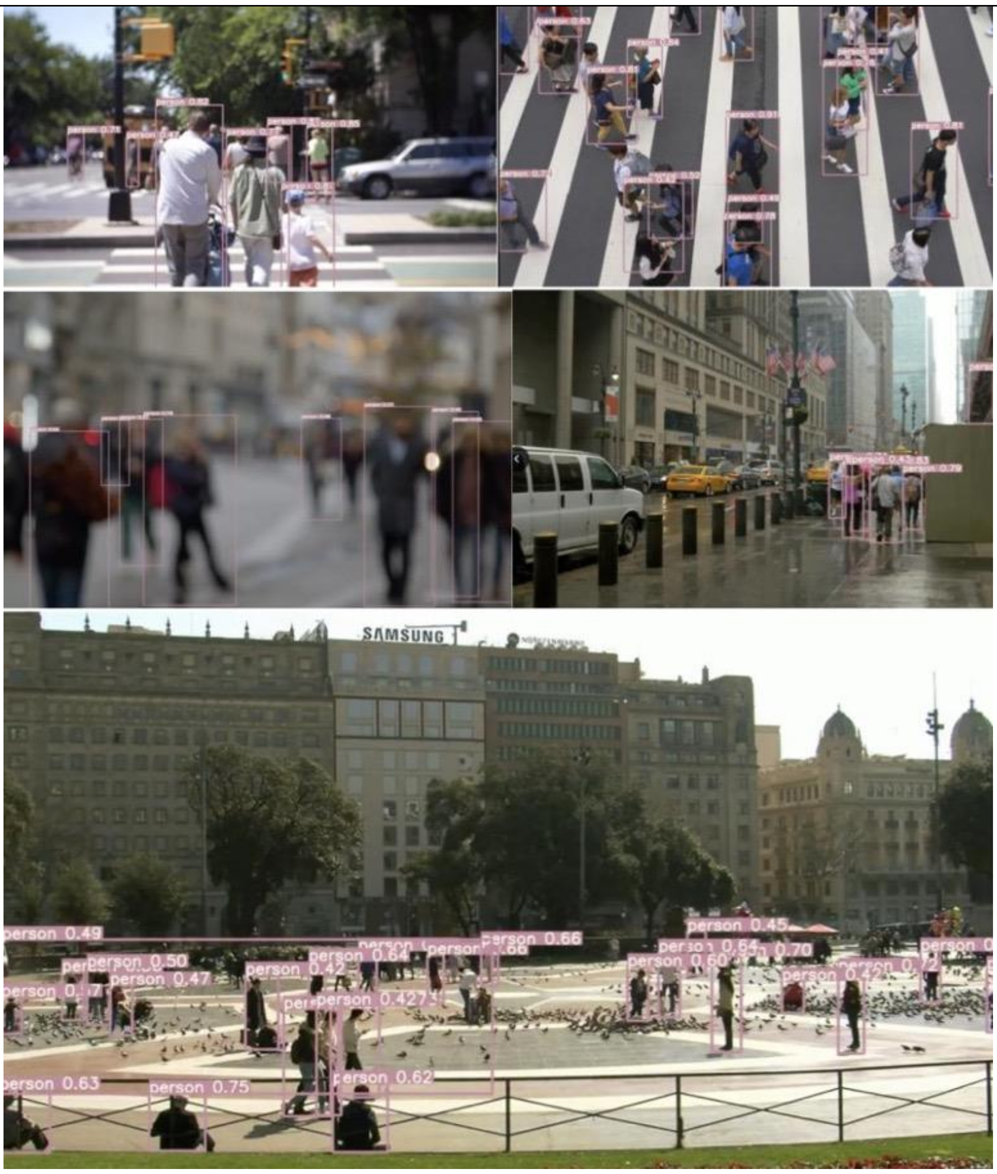


Figure 86. Video Mosaic of COCO on YOLOv4

At a first glance, it seems like this version of the model provides a better detection for objects in long range (see the fourth image) and it seems like it deals better with occlusions. For example, it is detecting the girl behind the obstacle in the middle of the last image, a person that YOLOv3 does not detect.

- **KITTI Dataset:**

As it is shown in the following mosaics, with YOLOv4, KITTI works even worse. It deals even worse with unknown poses and very badly with short range images.

It is clear that this new network architecture does not work well with out-of-format images and is less flexible than the previous version, which was lighter. In fact, far from perfect, with version 3, we could find some uses for the KITTI model, uses that are non-existent for the YOLOv4 version.

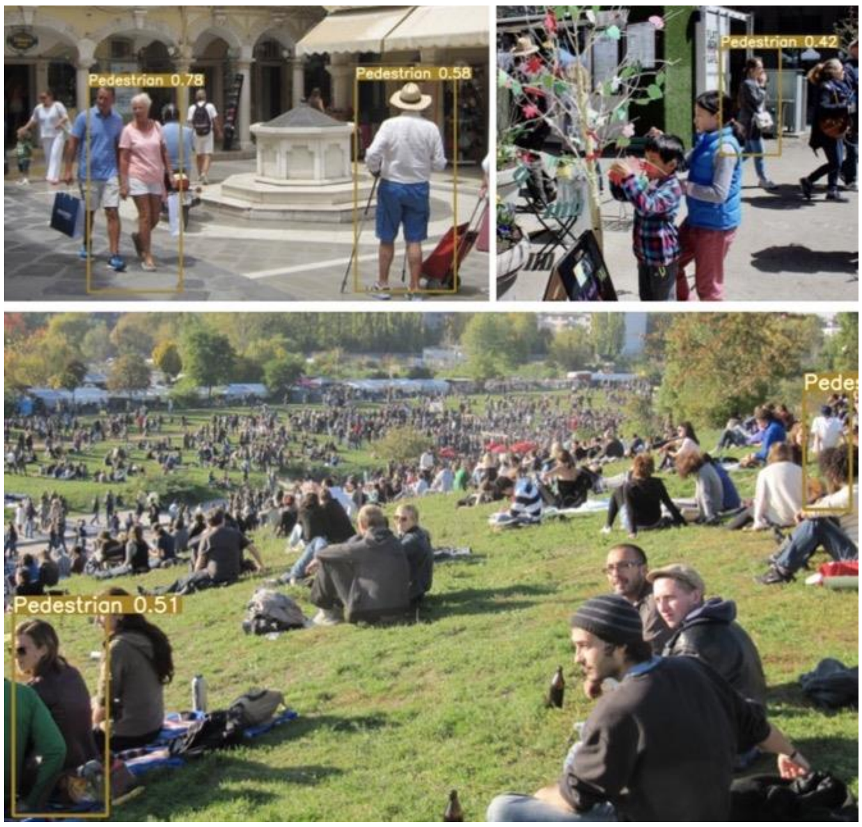


Figure 87. Image Mosaic of KITTI on YOLOv4

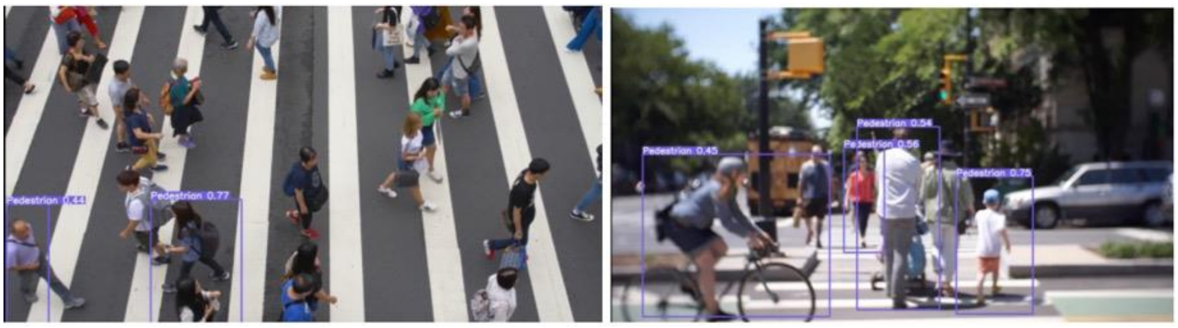


Figure 88. Video Mosaic of KITTI on YOLOv4

- **CrowdHuman Dataset:**

In light of the following captures, it is safe to say that with CrowdHuman, the best results are provided among the three datasets. However, it seems like it deals worse with long range camera action than the YOLOv3 version, which is kind of a problem.

In conclusion, I believe that for using the CrowdHuman dataset, there is the probability that using YOLOv3 as the model we can achieve a better result than using a heavier model as YOLOv4.

Let's not forget that inference time is higher in this model since it is heavier.

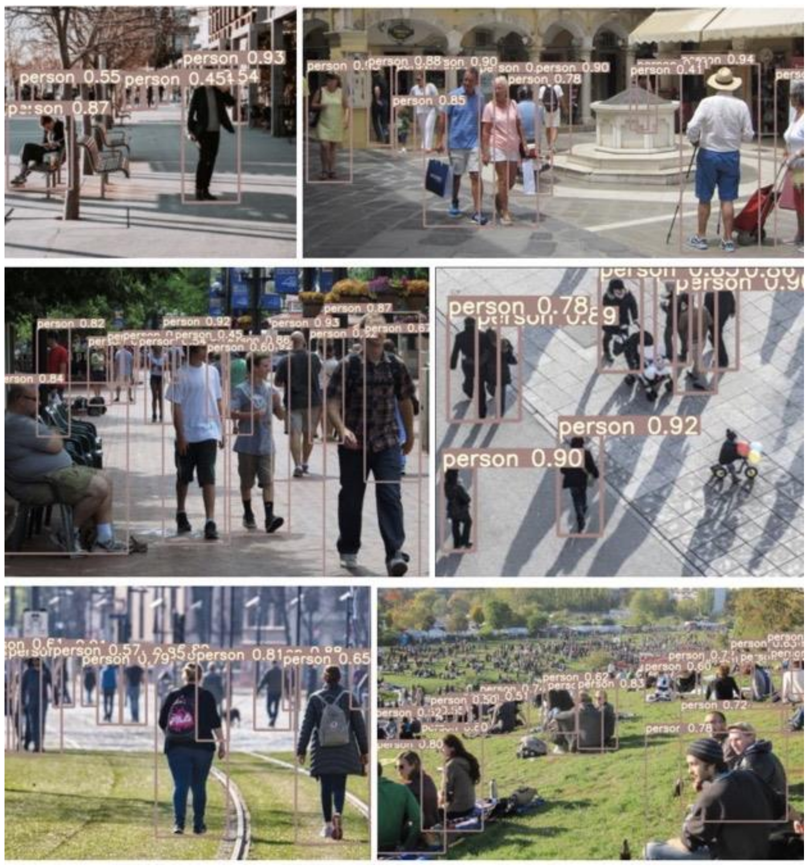


Figure 89. Image Mosaic of CrowdHuman on YOLOv4

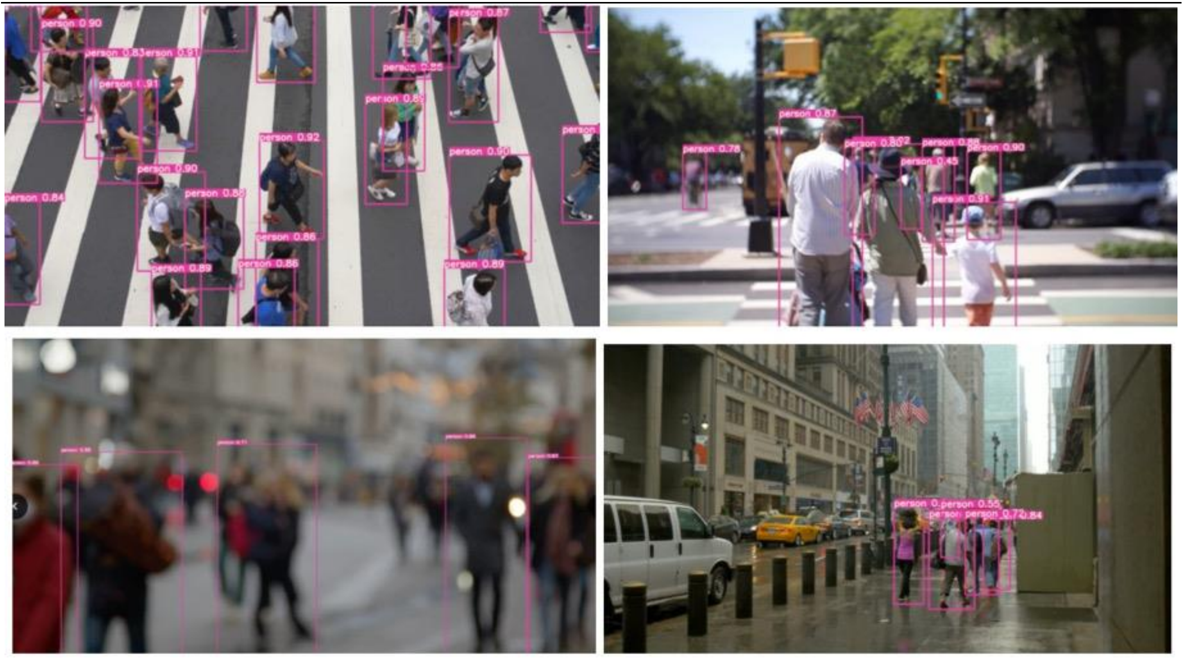


Figure 90. Video Mosaic of CrowdHuman with YOLOv4

5.3.4 YOLOv5

YOLOv5 is the newest "addition" to the YOLO family. However, there is some controversy about it and it´s naming. First of all, the developer and creator of YOLOv5 is not one of the authors of the original YOLO papers. In fact, there is no paper on YOLOv5 just yet, even though it was released as a product in June 2020. YOLOv5 was released by Glenn Jocher, CEO of Ultralytics[45] as an Ultralytics product. There is some controversy not only because of the different source that it comes from but on how it is implemented.

The network is implemented completely native in PyTorch, which actually is very useful for this project. By being implemented natively in PyTorch, it is, according to the results proposed in the GitHub repo [46], is much faster in inference and training time as the previous versions. By being implemented in PyTorch natively, it does not implement and parses the darknet framework used in the previous two versions. If we recall the .cfg file showed earlier, it is a 600 lines file for YOLOv3 and 1200 lines file for YOLOv4. When doing inference, that file is parsed to convert it into the model. Although it makes the framework very flexible, the complete code is much heavier. This is because Darknet was never thought to be efficient in production but more in terms of flexibility. The models I already trained are implemented like this, which is already a translation from Darknet to PyTorch since the original Darknet framework is written in C. With that translation I achieved the results outlined above, so with this new native implementation I expect a better result.

[45] YOLOv5 🚀 and Vision AI ⭐ (ultralytics.com)
[46] GitHub - ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite

While in training it is supposed to be faster, but not incredibly faster, according to their repo, for inference it is supposed to be more than two times faster than the previous version of YOLOv4. That is something very powerful for this type of task since we have this real-time constraint that is the driver that is going to decide which model to use.

As a side note, the weights of YOLOv5 are way smaller than the YOLOv3 and YOLOv4 versions, that were around 250 MB.The larger model I tried with YOLOv5, its weights are no more than 70 MB. This is important because they need to be loaded when doing inference and it is much faster to load a 70MB than a 250MB file.

In **Annex III: Darknet Vs. Ultralytics Approach**, there is the comparison between the Darknet scripts and parsing and how it is implemented in YOLOv5.

So far, YOLOv5 looks like a much better improvement than YOLOv4, but is it that different from the previous versions technically or is it just different because of how it is implemented?

Truth is that Ultralytics started to develop upgrades in the YOLOv3 version before YOLOv4 was released. Most of the improvements made were the same as what YOLOv4 paper proposed. As mentioned, YOLOv4 still followed the Darknet framework so, to avoid collisions, Ultralytics renamed their improvements YOLOv5, implemented natively on PyTorch. So, in summary, YOLOv5 is extremely similar in network architecture as YOLOv4, in fact, Glenn Jocher, the author, is credited in the YOLOv4 paper for its data augmentation development [43].

Essentially, its architecture follows the same pattern as the previous versions. It starts with a backbone based on a CSPDarknet53 network that allows to fix the vanishing gradient problem. Following the backbone is the neck of the algorithm. Just like YOLOv4, it uses a PANet network. It´s true that, as the paper says, other architectures can be implemented in this section but, YOLOv5 only focuses on the Path Aggregation Network to aggregate the features at different scales and layers of the network.

5.3.4.1 YOLOv5 - Training, Performance and Results

Due to lack of time, different YOLOv5 models will be trained by using transfer learning. Ultralytics provide different pretrained YOLOv5 models to use. They are different sizes, and transfer learning is going to be implemented with a couple of them. They are different sized models so I expect that the smallest one will provide a lower inference time and, consequently, a lower accuracy.

These models are pretrained on the COCO dataset, which is very useful for this project because there is no need to change any of the hyperparameters to train again on the COCO dataset used previously with only people. When training on CrowdHuman the hyperparameters were evolved to fit them into CrowdHuman characteristics more properly.

These models weren´t trained on the KITTI dataset, it is very clear that for this task, there is no use for the models trained on KITTI.

- **COCO Dataset:**

As it is shown in the graphs below, accuracy is significantly improved when using YOLOv5 model on the COCO dataset, it more stable and it jumps to more than 0.6 mAP using the smaller pretrained model and to more than 0.7 mAP using the medium-sized pretrained model. Let's take into consideration that this COCO dataset is still unbalanced as it was with the previous models. When using the larger model, it´s accuracy is the best that was reached in this project.

¿Why is the accuracy higher than YOLOv4 if the architecture is almost identical? For the YOLOv4 model trained them from scratch, the parameters weren´t set beforehand, and a pretrained model that had already proven to be very successful was not used. Adding to the mix the fact that the GPU used for the YOLOv4 models, and the GPU used for the pre-trained YOLOv5 models differ significantly in favour of the one used for YOLOv5 in terms of performance and processing capacity, that's the reason behind the difference in accuracy between both models.

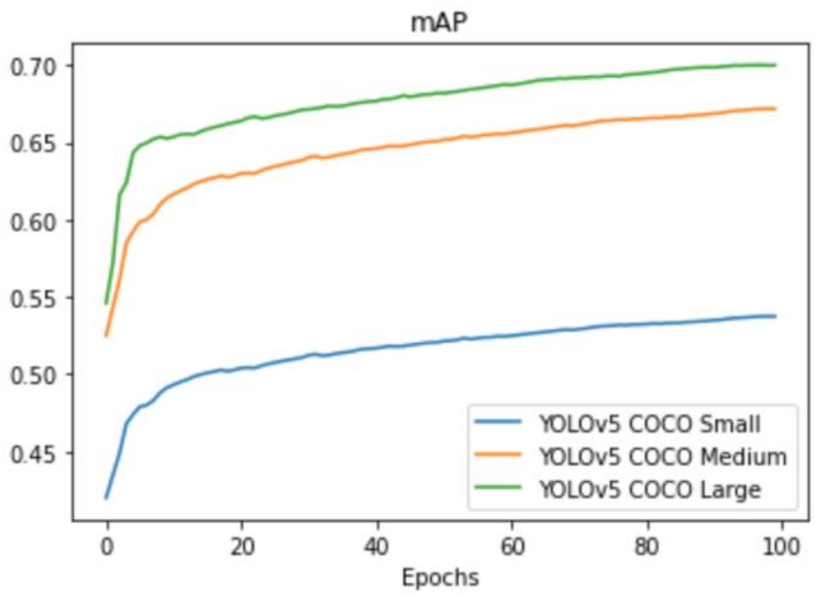


Figure 91. mAPs of the different YOLOv5 models on COCO

As seen in the above figure, both the medium and small versions grow in a similar manner from the pretrained weights, and the large model increases quicker. However, logically the larger model has more accuracy than the other two, but it is necessary to check that the inference time is not too high for that model.

As a curiosity, even the smaller model has more training accuracy than the fully trained model of YOLOv3 and YOLOv4.

- **CrowdHuman Dataset:**

Following COCO, three different versions of YOLOv5 were tried with the CrowdHuman dataset. To do this, it was necessary to adapt the hyperparameters from COCO to CrowdHuman. To do this, it was necessary to do some tryouts only for a couple epochs with different hyperparameters and see which ones provide better results. Of course, this is done automatically by the script. This allows the pretrained models to adapt to the characteristics of the CrowdHuman dataset before training on them. After performing the training for 100 epochs with the three versions, the results are the following.

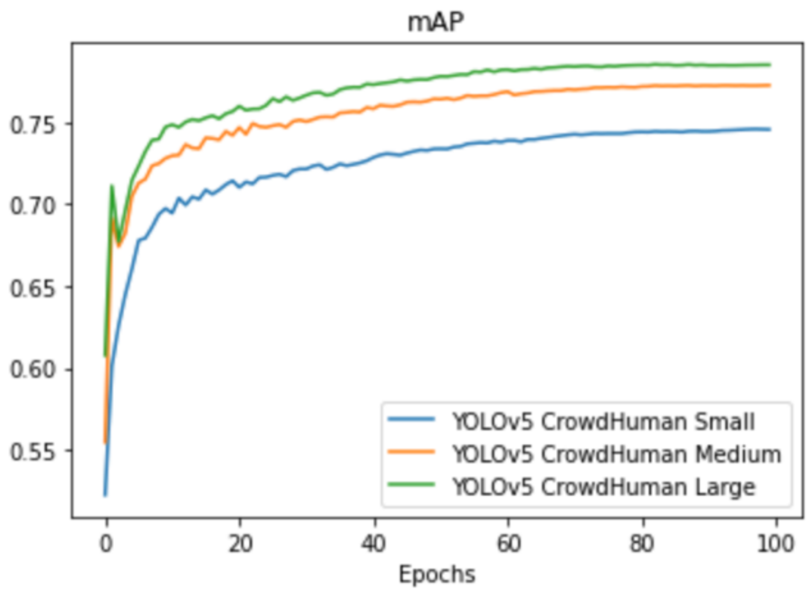


Figure 92. mAP for YOLOv5 on CrowdHuman dataset

Well, the results are very good, in fact, the smallest version provides better accuracy than the largest model with the COCO dataset. With that in mind, it is obvious that the model chosen will be one of these, since it will have less inference time and more accuracy than the COCO models.

To do a quick recall comparison to YOLOv3 models the below graphics show the recall values. As shown, in terms of recall, the model is a little more unstable but more precise.

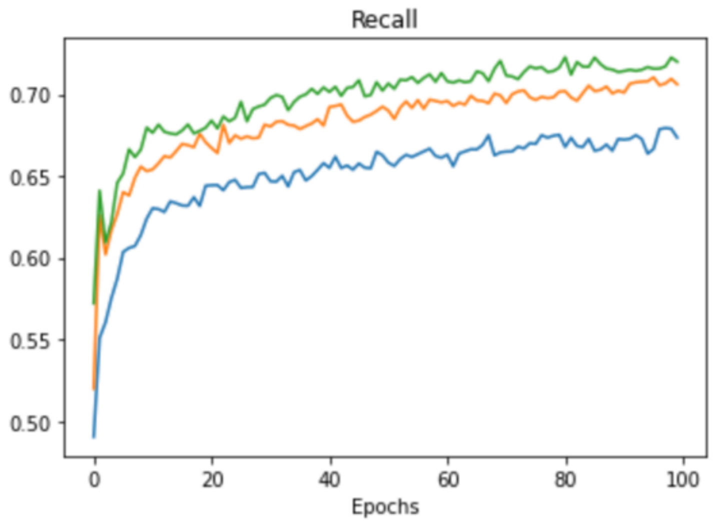


Figure 93. Recall for YOLOv5 on CrowdHuman dataset

5.3.4.2 YOLOv5 - Inference and Performance

- **COCO Dataset:**

As seen in the images below, the three different models of YOLOv5 provide similar results, visually, despite their differences in training accuracy. In terms of inference time, which a full comparison later, the YOLOv5 smaller model can do inference at more or less 10ms per frame, the medium sized model at 17ms per frame and the larger model at around 24ms per frame. Both the medium sized model and the smaller model have inference times lower than YOLOv3 model and the larger model have an inference time similar to YOLOv3 model. Having a similar accuracy in the results, the inference time is probably the most important metric to take into consideration later when I decide which model to use.

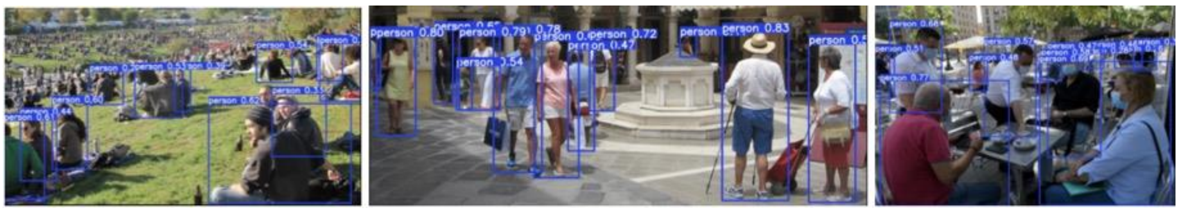


Figure 94. Small version of YOLOv5 on COCO

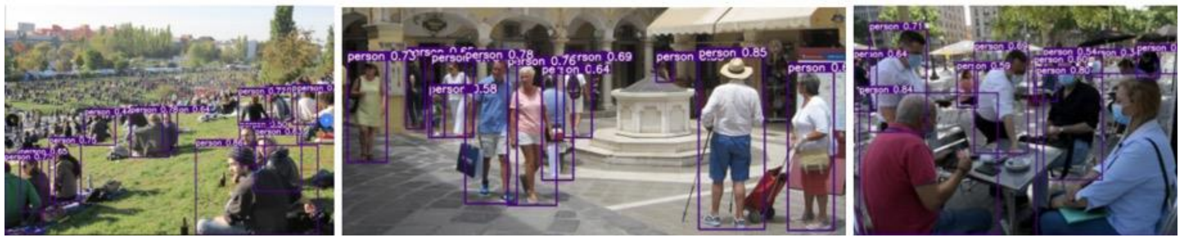


Figure 95. Medium version of YOLOv5 on COCO

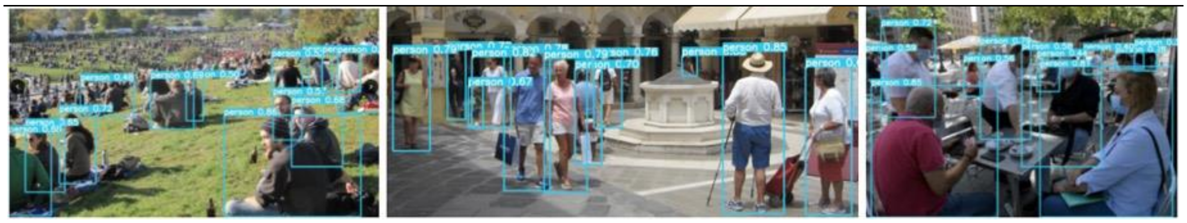


Figure 96. Large version of YOLOv5 on COCO

Visually, it doesn't seem to be much different between the three models. Later, I will provide a full comparison and check in detail but, the results are quite similar. The only difference is the inference time which is considerably lower with YOLOv5 even compared to YOLOv3.

- **CrowdHuman Dataset:**

The inference results provided are on the same three images as the previous sections. Again, all the results on all images and videos are on the link provided in **Annex II**.

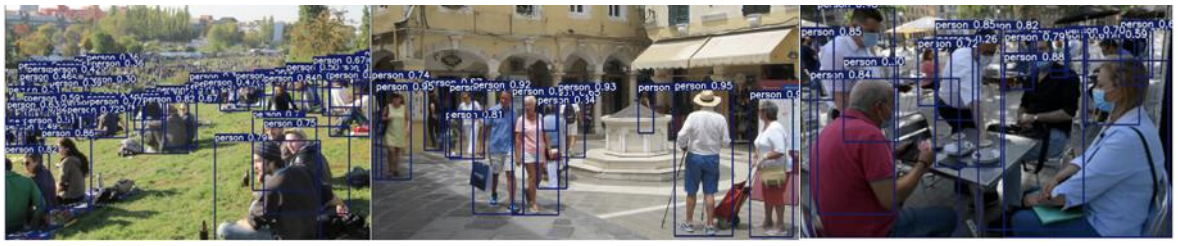


Figure 97. Small version of YOLOv5 on CrowdHuman

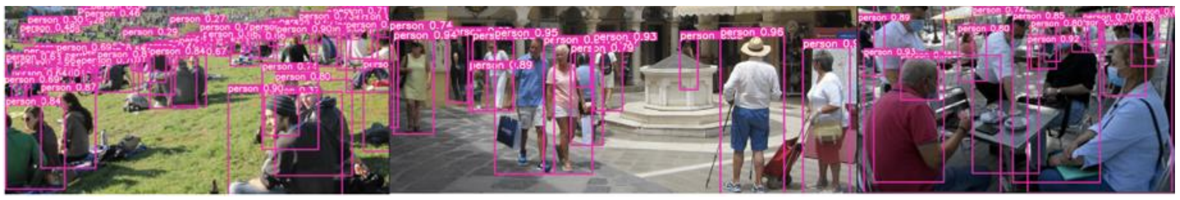


Figure 98. Medium version of YOLOv5 on CrowdHuman

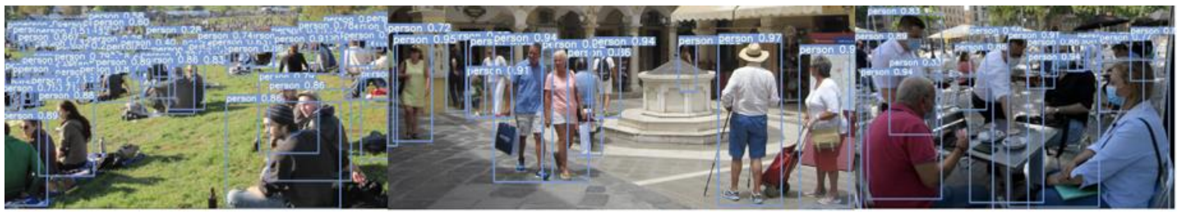
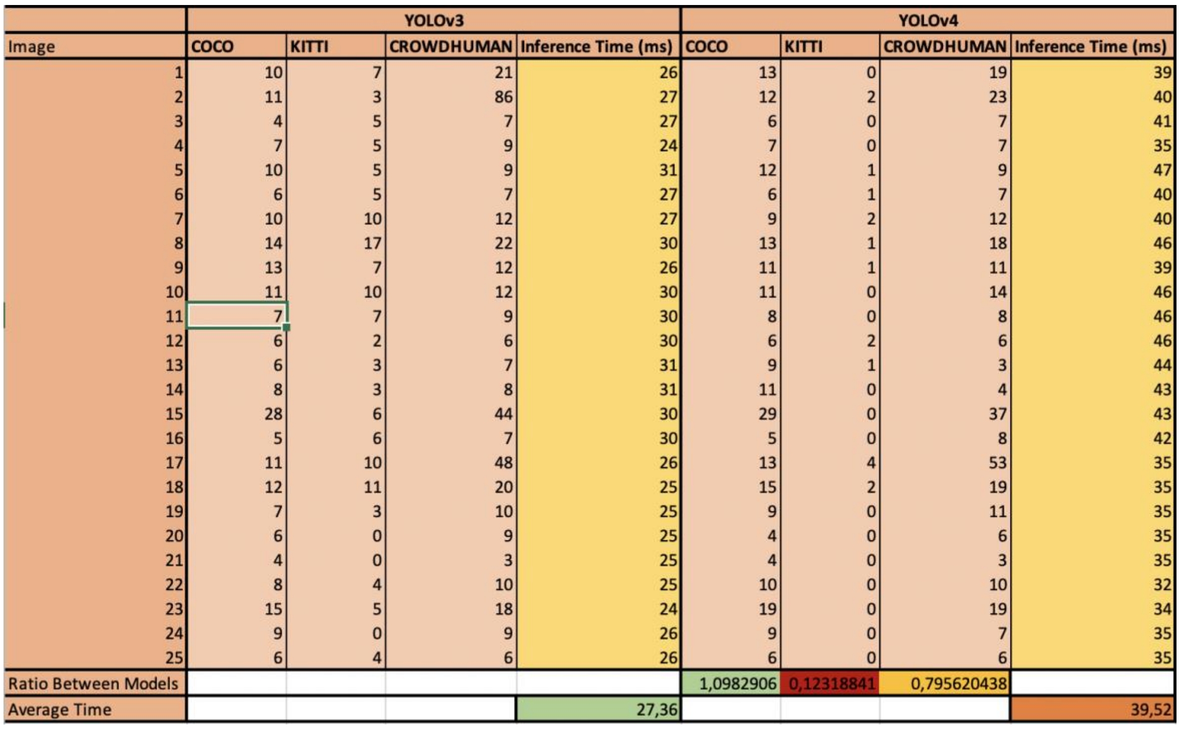


Figure 99. Large version of YOLOv5 on CrowdHuman

As seen in the above, the more accurate model is the large one, logically, but as shown in the following section, and taking into consideration that the priority is real time, probably the medium sized model is the correct option to choose.

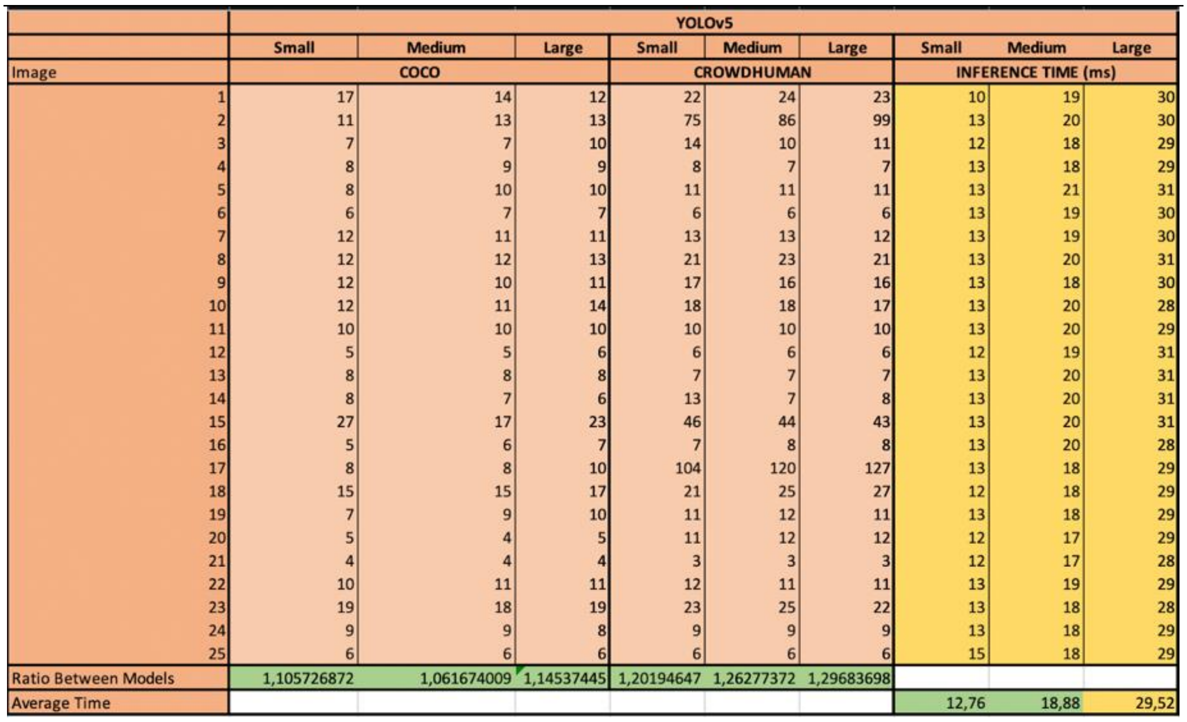
5.4 COMPARISON OF RESULTS OF THE MODELS & CONCLUSIONS

In the tables below, you can see, for the 25 images where the models have been tested, the number of detections made and the inference time in milliseconds. For all the models, the images are exactly the same and they depict people in public spaces like parks, restaurants or malls. I have used the majority of the images in the previous sections when displaying the results of the models. The ratio between the models measures the percentage of detections done by that model compared to the YOLOv3 model on that dataset. This applies for both of the tables below.



	YOLOv3				YOLOv4			
Image	COCO	KITTI	CROWDHUMAN	Inference Time (ms)	COCO	KITTI	CROWDHUMAN	Inference Time (ms)
1	10	7	21	26	13	0	19	39
2	11	3	86	27	12	2	23	40
3	4	5	7	27	6	0	7	41
4	7	5	9	24	7	0	7	35
5	10	5	9	31	12	1	9	47
6	6	5	7	27	6	1	7	40
7	10	10	12	27	9	2	12	40
8	14	17	22	30	13	1	18	46
9	13	7	12	26	11	1	11	39
10	11	10	12	30	11	0	14	46
11	7	7	9	30	8	0	8	46
12	6	2	6	30	6	2	6	46
13	6	3	7	31	9	1	3	44
14	8	3	8	31	11	0	4	43
15	28	6	44	30	29	0	37	43
16	5	6	7	30	5	0	8	42
17	11	10	48	26	13	4	53	35
18	12	11	20	25	15	2	19	35
19	7	3	10	25	9	0	11	35
20	6	0	9	25	4	0	6	35
21	4	0	3	25	4	0	3	35
22	8	4	10	25	10	0	10	32
23	15	5	18	24	19	0	19	34
24	9	0	9	26	9	0	7	35
25	6	4	6	26	6	0	6	35
Ratio Between Models					1,0982906	0,12318841	0,795620438	
Average Time				27,36				39,52

Table 3: Comparison of Inference Time and Number of Detections (YOLOv3 & YOLOv4)



	YOLOv5								
	Small	Medium	Large	Small	Medium	Large	Small	Medium	Large
Image	COCO			CROWDHUMAN			INFERENCE TIME (ms)		
1	17	14	12	22	24	23	10	19	30
2	11	13	13	75	86	99	13	20	30
3	7	7	10	14	10	11	12	18	29
4	8	9	9	8	7	7	13	18	29
5	8	10	10	11	11	11	13	21	31
6	6	7	7	6	6	6	13	19	30
7	12	11	11	13	13	12	13	19	30
8	12	12	13	21	23	21	13	20	31
9	12	10	11	17	16	16	13	18	30
10	12	11	14	18	18	17	13	20	28
11	10	10	10	10	10	10	13	20	29
12	5	5	6	6	6	6	12	19	31
13	8	8	8	7	7	7	13	20	31
14	8	7	6	13	7	8	13	20	31
15	27	17	23	46	44	43	13	20	31
16	5	6	7	7	8	8	13	20	28
17	8	8	10	104	120	127	13	18	29
18	15	15	17	21	25	27	12	18	29
19	7	9	10	11	12	11	13	18	29
20	5	4	5	11	12	12	12	17	29
21	4	4	4	3	3	3	12	17	28
22	10	11	11	12	11	11	13	19	29
23	19	18	19	23	25	22	13	18	28
24	9	9	8	9	9	9	13	18	29
25	6	6	6	6	6	6	15	18	29
Ratio Between Models	1,105726872	1,061674009	1,14537445	1,20194647	1,26277372	1,29683698			
Average Time							12,76	18,88	29,52

Table 4: Comparison of Inference Time and Number of Detections (YOLOv5)

As seen on the first table, YOLOv3 produces very good results both in number of detections and inference time. When comparing the datasets, it is seen that for the same image, CrowdHuman trained models are able to detect more people than the models trained on the other datasets, especially when dealing with long range shots and complicated perspectives. In terms of performance, YOLOv3 outperforms considerably YOLOv4 in inference time. In terms of number of detections, it is true that YOLOv4 is able to detect some more people than YOLOv3 when using COCO, but, if using KITTI, that accuracy is ridiculously low. The case of CrowdHuman is confusing. It shows that YOLOv4 detects 80% of the people that the YOLOv3 model can detect but, if you take a closer look, all of those detections are lost on the same image (image number 2), which is the image shown below**.** Judging from this sole image, it shows that YOLOv4 lacks performance when dealing with long range, but it deals a little better with occlusions, as shown before. If we do not count that problematic image, the ratio of detections jumps to more or less 0.95, which is not that bad. However, it seems that the correct model to choose from those is clearly YOLOv3, it is a little bit less accurate in training, but it is able to detect more people, at least in these experiments, and it is considerably faster. And being more precise, the correct model is trained on CrowdHuman.

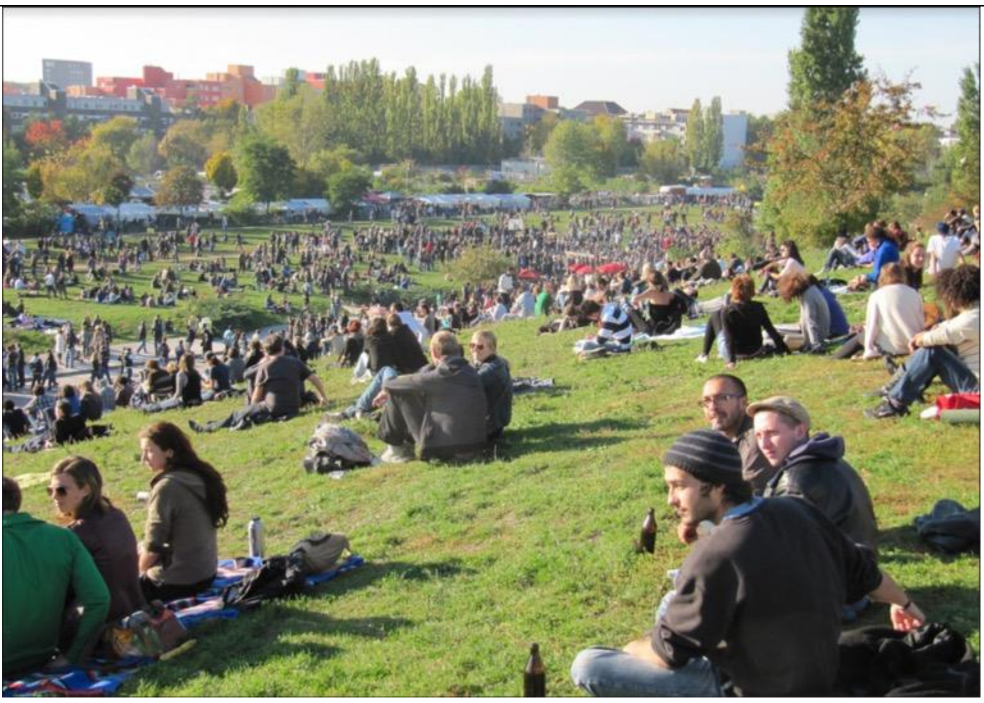


Figure 100. Conflicting image

But there is also YOLOv5, which is even faster when using the appropriate weights. All the metrics are considerably better than YOLOv3 and, of course, YOLOv4. In terms of inference time, YOLOv5 cuts the inference time in less than half for the smaller model, reduces by 10ms the time when using the medium sized model and when using the larger model, the inference time is the same as YOLOv3, but with a lot more accuracy. The best performing YOLOv3 model is still going to be used, which is the CrowdHuman one, for the tracking to see how it performs, but YOLOv5 is much better.

In terms of accuracy there is no question about which model is better, just by looking at the mAP curves and inference YOLOv5 is way better, specially the CrowdHuman model that way more people in an image, especially in frames with occlusions and long-range shots (4 times better, check images 2 and 17 on the tables), as seen in the inference examples. Comparing the YOLOv5 CrowdHuman models to the YOLOv3 ones, they can detect, on average, 125% of the people the YOLOv3 model can.

Considering these results, I decided to move on with two models. As said earlier, I want to try a YOLOv3 model on the tracking, so I am going to pick the YOLOv3 model of CrowdHuman as it is the best performing one. Of course, a YOLOv5 model needs to advance, and I am going to choose again, logically, a CrowdHuman one. I decided to move on with the medium sized one. In terms of performance there is not much difference to the large one and it is more than good enough in terms of mAP and number of detections. The inference time is also a very important reason, this model allows me to reduce inference time by 10ms from the YOLOv3 model and 12ms from the larger YOLOv5 model.

Regarding the COCO dataset, for general object detection is the best possible dataset, but for more specific tasks like this, it is way better to use a more specific dataset like CrowdHuman.

CHAPTER 6. TRACKING

6.1 INTRODUCTION TO TRACKING

The next stage in the project is to provide some tracking within the frames to make sure that we do not lose the identification of the people that the previous algorithm detects. This can be seen as a "safety" measure because, truth is that it is possible to feed an object detector directly into the distance estimator, if the detector is rock solid. However, adding a tracking mechanism in between those stages, allows the algorithm to "memorize" previous detections for a certain amount of time in case it loses a detection in a given frame, but the same person appears again in the following one. It is a very efficient method to deal with occlusions. The main drawback is that the inference time increases, so it is very important to choose, when trying different detectors, the one with the lower inference time while maintaining a sufficient accuracy. A key example is to use YOLOv3 instead of YOLOv4, which has a higher accuracy but almost doubles inference time.

In this project, the approach will be tracking-by-detection. This approach takes consecutive frames and tracks detections through them. This is different for example than re-identification, which might sound similar but it isn´t. That approach does not use consecutive frames but tries to match a detection in a given image with a previous detection already done. Re-identification is not a real-time approach. Since our input will be a video feed, tracking-by-detection is the obvious choice for the task. There are other approaches such as graph-based algorithms that are more accurate, but the inference time is much higher and therefore, they are not suitable for real-time applications. One of the reasons for this is that they need to consider information from the future, as they normally use Shortest Path algorithms, which need that kind of information [45].

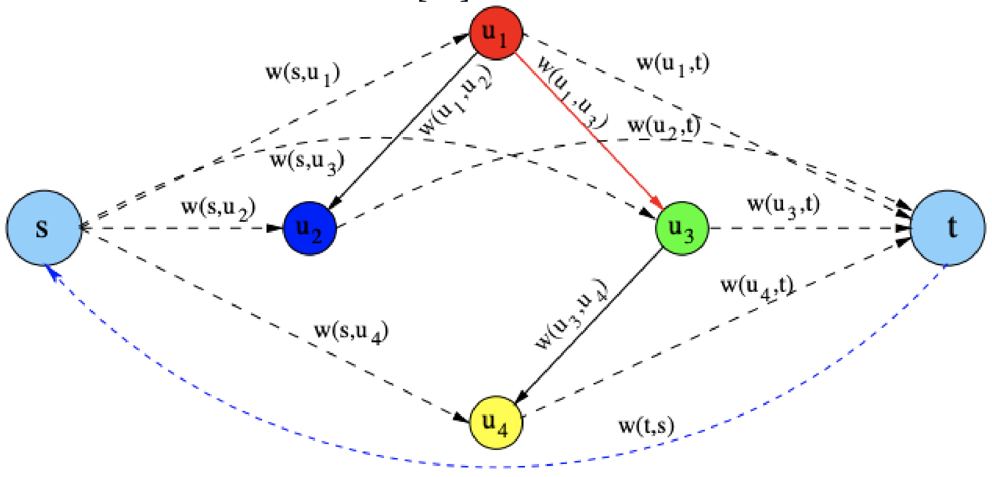


Figure 101. Shortest Path Algorithm Structure [45]

The complicated part is to match detections within frames. To do this the algorithms use three different metrics or techniques. Given two bounding boxes (detections) in consecutive frames, they can match them using three different types of information. The first one is appearance. Appearance is measured image wide. It is the result of a comparison between the output vectors of the consecutives images after the network. The second measure is the difference between the centers of the bounding boxes. This is done for each bounding box. It is necessary to assume that between frames (especially if the algorithm can work in a high fps near real-time), the people detected cannot move from one corner to another so, the closest centers will probably be the ones corresponding to the same detection. Finally, the same principle is applied to compute the difference between the size of the bounding boxes. Normally, the size of the bounding box will be the same within frames. In summary, after processing the similarity of output vectors, if the centers are positioned in very similar locations and the difference in size is very low, those bounding boxes probably belong to the same detection.

Dealing with occlusions is another complication but, there is a difference between occlusions between same class detections and occlusions between different objects. The occlusions between different objects are much more manageable, and it is done by using the three methods explained before. For example, a car that occludes a person may have a very similar center of the bounding box as the person in the previous frame, however, the size of the bounding box is probably not going to be close enough to confuse the detections. If same class detections occur, dealing with them is more difficult. The second and third metric become almost insignificant. That is because, if a person stands in front of another person, and the camera is static, it is very probable that the centers and the size will be very similar, except in cases of people very physically different between them. The key to solve it relies mostly on the detector and the network and how good and distinct are the output vectors that they provide.

What most tracking algorithms use is an internal “movement predictor”. What this does is that they record and memorize the movement information of detections in previous frames, this can be location, direction, shifts in direction and velocity, among others. This is done by using algorithms such as the Hungarian algorithm or Kalman filters, that both try to match old tracks, that is what that movement information is called to new detections. For this project I am going to use an algorithm based on Kalman filters called DeepSORT. They are both explained in the State of the Art but, essentially, the Hungarian algorithm is capable of matching the same bounding boxes in consecutive frames and Kalman filters can predict and estimate the future position of said bounding box. Using both in conjunction is necessary in most tracking algorithms. The Hungarian algorithm computes the previously explained IoU between current and past bounding boxes, a shape score between the bounding boxes and even a convolutional cost, that means that after applying a convolutional layer to the images, if the features are the same, the detections are the same. IoU is the most commonly used. After a match has been made, it uses a predict and update functions. These functions update two matrices, one called mean, that contains the velocities and another that computes the uncertainty. The uncertainty matrix is changed in the update function to better predict positions.

The main challenges are the occlusions and the missed detections, that essentially are occlusions also. That is why it is so important to remember the movement information of previous frames.

Traditional methods that implement tracking in computer vision are Meanshift and Optical Flow. Meanshift is mainly used in clustering problems, therefore unsupervised. Instead of computing the centroids like a normal K-Means clustering does, it uses a weighted average that gives more importance to the points closer to the mean [46][47]. Meanshift also determines the number of clusters suitable for each dataset [47]. This makes the algorithm use more time, therefore making Meanshift not suitable for real-time applications. In terms of tracking, Meanshift generates a color histogram of the content of the bounding box and looks for the closest match on the consecutive frames. Optical Flow is different. In fact, it is not a pure tracking-by-detection method. It does not need previous features from the detected objects, necessarily. The object is tracked measuring the pixel brightness variations in the frames [46]. As mentioned earlier on the State-of-the-Art, this method focuses on finding a displacement vector to track along the frames. The tracking using these two methods is good, but is very computationally complex, prone to noise and tracks can be lost due to occlusion.

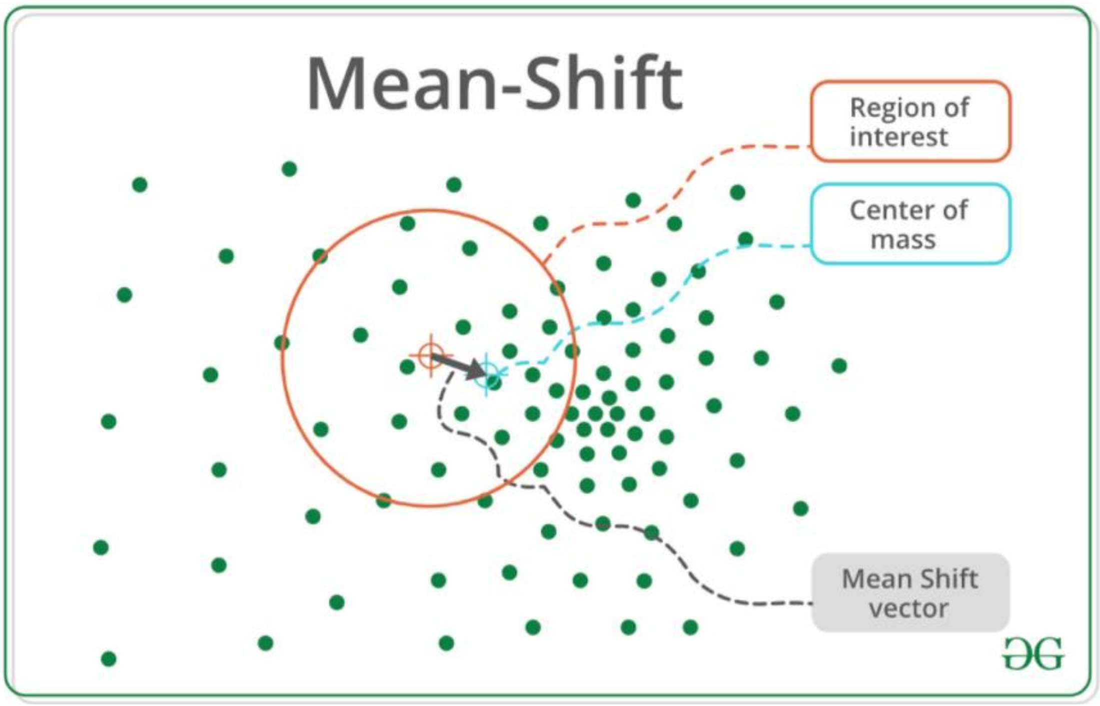


Figure 102. Meanshift Approach [47]

More modern approaches are already in the field of deep learning. There are two that got more recognition from the deep learning community. They are Deep Regression Networks and, the other, which is somewhat familiar, ROLO (Recurrent YOLO). Deep Regression Network models are trained on videos, and they simply have to track an object from the image crop. They use regression to go from the current image detection of the object back to the object detected in the previous image [49]. After detecting an object in a frame, based on the region it was detected, it defines a search region on that region on the previous image. The architecture can be seen in the following image.

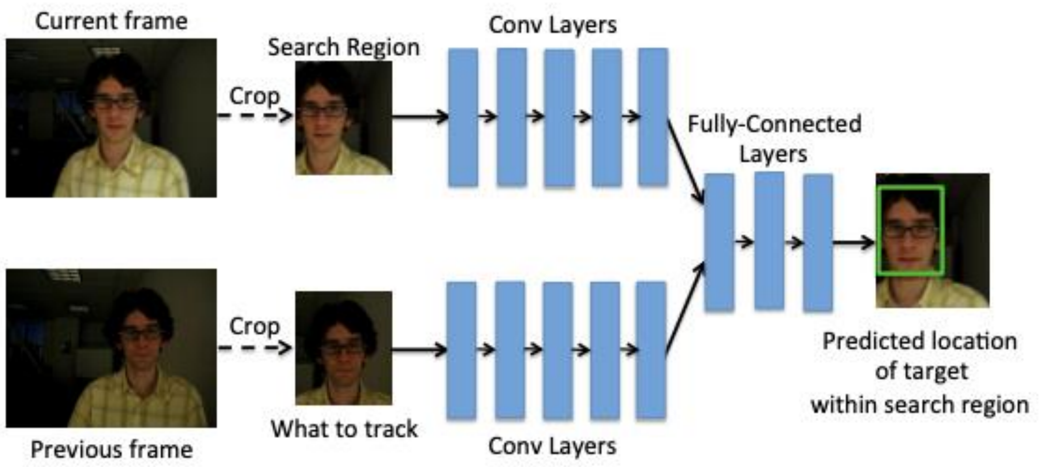


Figure 103. Deep Regression Network architecture [49]

ROLO implements the extensively explained YOLO architecture and attaches a LSTM network at the end. All the feature information extracted from the feature extractor (Backbone + Neck) is passed to the LSTM, which provides the bounding box predictions and the time and space information. It seems that the logical path for the project is to use ROLO but, unfortunately, ROLO is, for now, only contemplated for single object detection and struggles when it has to deal with data association [50]. For this project, it is, by definition, necessary to track multiple objects so that the distance between them can be estimated. That is why I decided to use DeepSORT and not use ROLO.

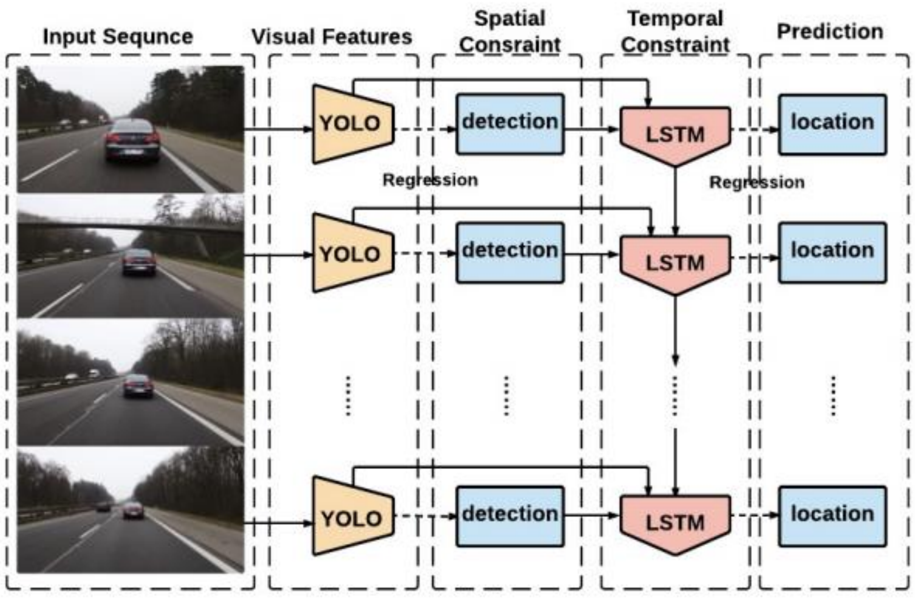


Figure 104. ROLO architecture [50]

6.2 DEEPSORT

The most popular tracking algorithm is DeepSORT. It is an evolution of the SORT (Simple Object Real Time Tracker) [51] algorithm, which was explained in the State-of-the-Art section. The main difference between the two algorithms is that DeepSORT replaces the Hungarian method association metric with a CNN network.

As mentioned before, DeepSORT relies on Kalman filters using the detections of the previous frames to determine a best guess of the position the bounding box is. Kalman filters work in a cyclical manner as the picture below depicts [48].

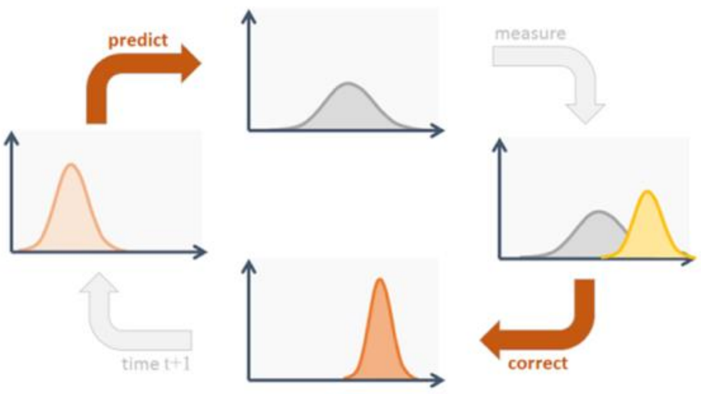


Figure 105. Kalman FIlters Workflow [48]

It is assumed that the velocity is constant, so, when a sudden acceleration occurs, the model struggles. To try to fix that, it generates the uncertainty matrix mentioned before and updates it every cycle when the predictions are not a match. Since by using a constant velocity model it is a linear approach, it is quite useful for our task. Us humans tend to move in a quasi-constant velocity, and in normal situations, it is very complicated for us to accelerate or stop so quickly that the model cannot update the weights accordingly, our physique does not allow it.

The Kalman filter generates not only the bounding boxes but also the ‘tracks. That is the information of the localization of the objects in the previous frame, the last steps that a person took, in our case. To make the algorithm work we need to associate the track to the new detection.

The authors of the paper [51], decided to use the Mahalanobis distance, which allows them to take into consideration the uncertainty, and work better with distributions. Remember, Kalman filters do not output two points were doing the Euclidean distance will suffice, the output is a distribution in matrix form. The formula of the Mahalanobis distance is below. The authors threshold the Mahalanobis distance metric at a 95% confidence interval.

$$\boldsymbol{d\,(1)(i,j) = (dj - yi) * TSi - 1 * (dj - yi)}[51]\ (3)$$

After applying those two methods (Kalman and Hungarian) to the frames in the SORT algorithm, Deep Learning does not seem to be that necessary, but, despite the effectiveness of the filters and the associations provided by the Hungarian algorithm, it still struggles with occlusions and different viewpoints (Nothing new at this point). Deep learning is used to fix this by introducing the appearance metric mentioned before. As explained before, the idea is to obtain a feature vector that accurately describes the images. After obtaining the feature vector the distance metric is updated like equation (4) suggests:

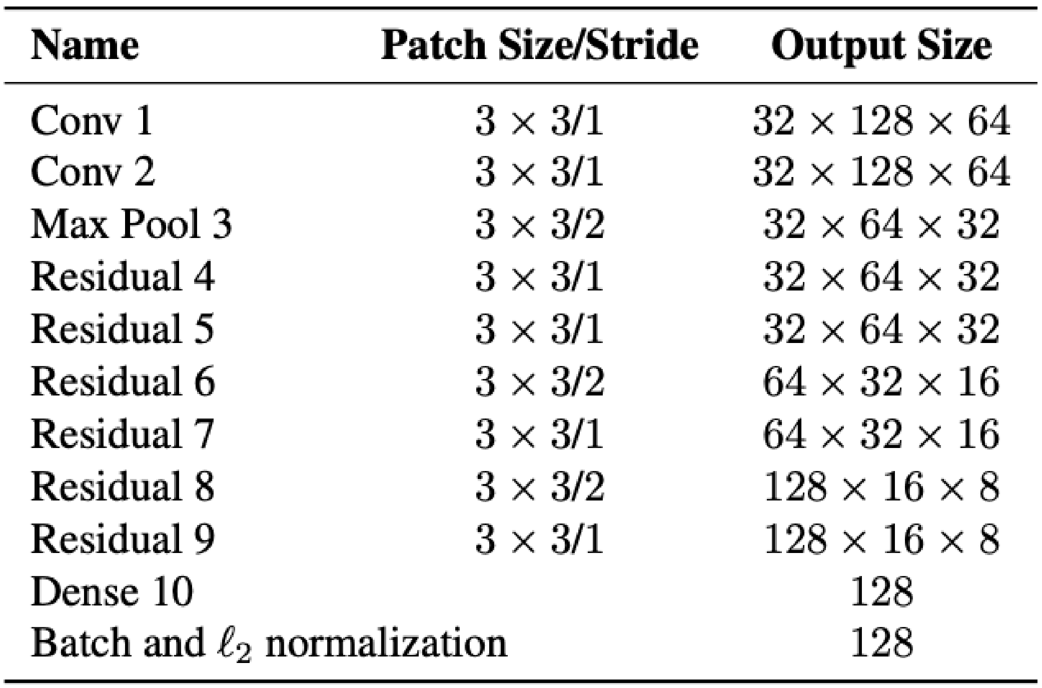
$$\boldsymbol{ci,j = \lambda\, d(1)(i,j) + (1 - \lambda) d\,(2)(i,j)}[51]\ (4)$$

In the equation (4), d(1) is the Mahalanobis vector used before, d(2) is the cosine distance between the feature vectors of the current and previous image and Lambda is the weighing factor. The authors [51] claim that the importance of d(2) is much more than d(1) that they were able to achieve good results using Lambda = 0, nullifying the Mahalanobis distance, when there is considerable camera motion, therefore more occlusions. Using this deep learning feature increased the power and accuracy of DeepSORT considerably (Before, it was just the SORT algorithm). To compare consecutive frames in terms of appearance descriptors, the algorithm takes the minimum cosine distance between the i-th track and the j-th detection [51], as seen in equation (5).

$$\boldsymbol{d\,(2)(i,j) = min\{1 - rj\, Tr\,(i)\, k \mid r\,(i)\, k \in Ri\}}[51]\ (5)$$

In summary, the Mahalanobis measure provides useful information for short term detection for bounding box localization while the appearance descriptor considers information that is very useful to recover predictions after some time.

This CNN part of the algorithm is relatively simple compared to what YOLO was, but it is key to DeepSORT success. The architecture of this CNN network is as follows [51]**.**



Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Figure 106. Convolutional Network Architecture in DeepSORT [51]

As it is shown, the architecture is simpler than YOLO, which helps to not push the inference time too high.

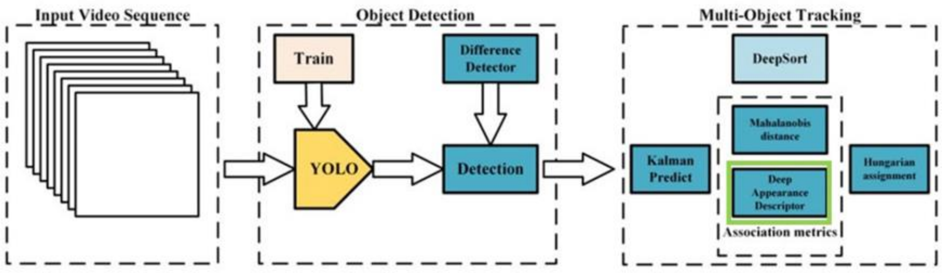
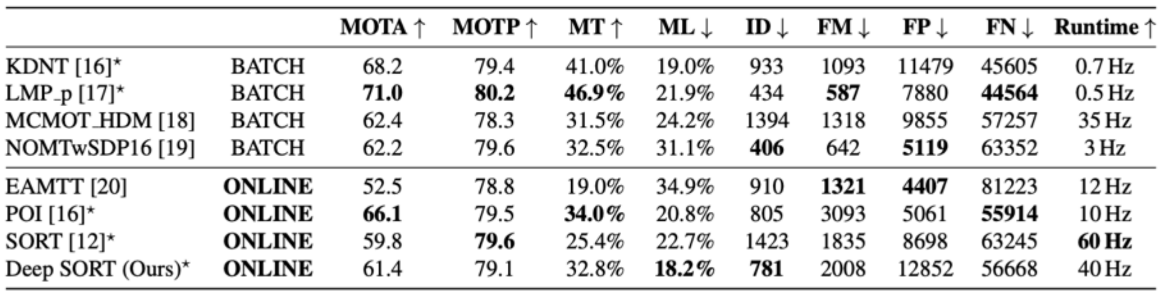


Figure 107. Full Architecture of DeepSORT[47]

47 https://augmentedstartups.medium.com/deepsort-deep-learning-applied-to-object-tracking-924f59f99104

6.2.1 DEEPSORT TRAINING

The authors in the paper evaluated the model using the MOT challenge [52]. This challenge is specifically designed to track objects in public locations, including people. The results obtained were the following:



		MOTA ↑	MOTP ↑	MT ↑	ML ↓	ID ↓	FM ↓	FP ↓	FN ↓	Runtime ↑
KDNT [16]*	BATCH	68.2	79.4	41.0%	19.0%	933	1093	11479	45605	0.7 Hz
LMP_p [17]*	BATCH	**71.0**	**80.2**	**46.9%**	21.9%	434	**587**	7880	**44564**	0.5 Hz
MCMOT_HDM [18]	BATCH	62.4	78.3	31.5%	24.2%	1394	1318	9855	57257	35 Hz
NOMTwSDP16 [19]	BATCH	62.2	79.6	32.5%	31.1%	**406**	642	**5119**	63352	3 Hz
EAMTT [20]	**ONLINE**	52.5	78.8	19.0%	34.9%	910	**1321**	**4407**	81223	12 Hz
POI [16]*	**ONLINE**	**66.1**	79.5	**34.0%**	20.8%	805	3093	5061	**55914**	10 Hz
SORT [12]*	**ONLINE**	59.8	**79.6**	25.4%	22.7%	1423	1835	8698	63245	**60 Hz**
Deep SORT (Ours)*	**ONLINE**	61.4	79.1	32.8%	**18.2%**	**781**	2008	12852	56668	40 Hz

Table 5: DeepSORT training metrics on MOT challenge [51]

Taking into consideration that these results are very good in a similar task, for this task was used the already trained model of DeepSORT and put after the detectors trained before (Just the ones selected to "advance"). Training DeepSORT includes training also the detectors so, since the detectors for this project are trained, it makes a lot of sense to use the already pretrained DeepSORT model and use for inference the detectors previously trained. The model provided by the authors allows to select different parameters for the inference such as IoU, Confidence and Age of Tracks, which is a feature of DeepSORT that allows the user to choose the frames you want to keep a lost tracks information for. This allows the model to not overload the tracks matrix and keep the model lighter. It works in a very simple way, whenever a track is lost, because of detectors mistake, occlusions etc., the user can set the number of frames he wants that track information to be kept so, in case the object appears again, the user can match it with the previous detection. For example, setting the age to 50 frames, if an object is lost at a given frame 'x', the track info for it will be kept until frame 'x+50', if the object appears again at frame 'x+25' it will be able to match the detections, but after frame 'x+50', the track information will be erased and if it appears again, it will be identified as a new detection. This is the tradeoff to be made between the need for real-time and the loss of precision, if the age is very high, the inference time will increase. Personally, for this task, I am choosing a 60 frames age initially and then checking the inference time to see if it works or if it needs to be reduced. For this case, if the frame is lost for more than 60 frames, at 30 fps is 2 seconds, it is considered it to be a detection that is going to appear either on a completely different location of the frame or it is not going to appear again at all after that time.

6.2.2 DEEPSORT INFERENCE

Since it is very complicated to show a result that is on video here, I will attach a link to the Google drive folders from each dataset and model after my thoughts and conclusions for each of them. However, a sequence mosaic of a video for each dataset will be provided to show a preview of how the algorithm works. In general terms, for both datasets the algorithm provides good results. To be able to draw a valid comparison, two models were previously chosen to go along with the projects, one with YOLOv3 and another one with YOLOv5. As explained before, the best model from the YOLOv3 versions is the one trained on CrowdHuman, so that is one of the models that advances. From the YOLOv5 ones**,** as said earlier, the best option is to continue with the medium sized model trained on CrowdHuman, is almost as accurate as the larger model and it is 12ms faster on average as it. It is also 10ms faster than the YOLOv3 model.

On the YOLOv3 model, you can see below two different sequenced videos, divided into frames. Taking a look at the IDs, it can be seen that they do not change even after occlusions. I know it is difficult to identify, but bear with me. All the links to a Drive folder with all the solutions are shared in **Annex II: Inference Results.**

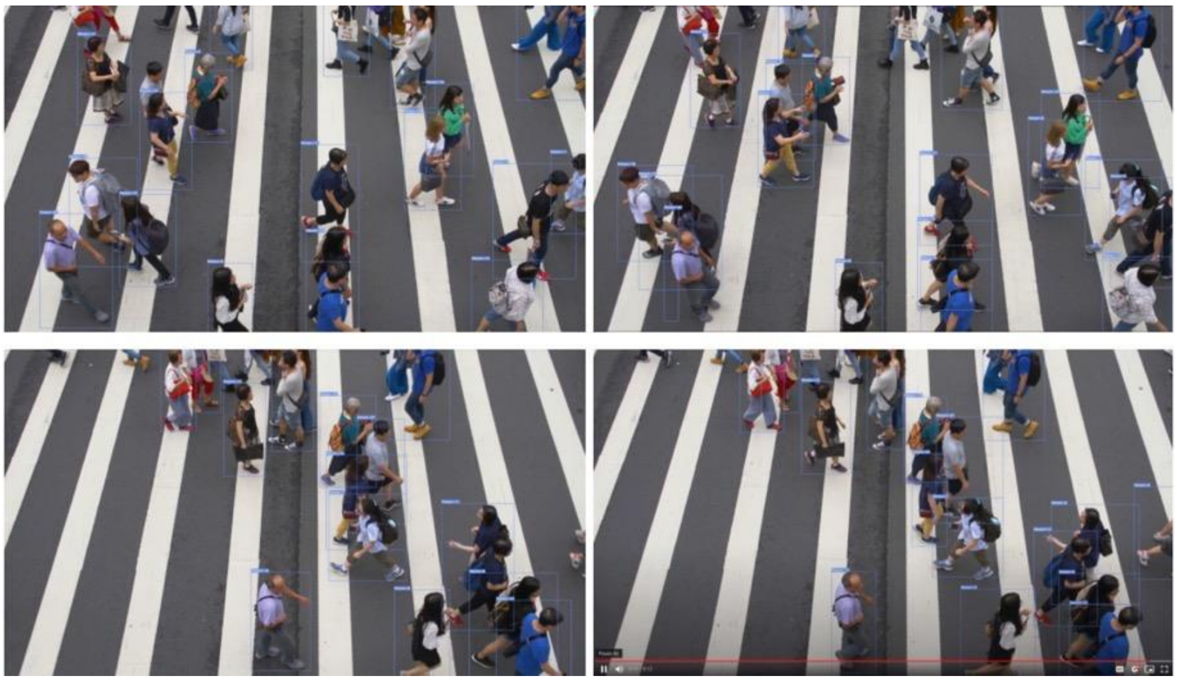


Figure 108. YOLOv3 + DeepSORT - Video 1

In the above sequence you can see very clearly how the IDs are not lost at all. The perspective here helps a lot, since the algorithm is able to recover and "see" the tracks very easily. In the following sequence, you can see that for most IDs, the tracking is done almost perfectly (For example, ID9) but it has some problems. Taking a look at ID75, before the crossing of the bike it is assigned to the man with the woman and kid. However, after the bike occludes them, ID75 is assigned to the woman coming in the exact same trajectory as the previous man. Truth is, it is perfectly understandable that the tracks are mixed up in that case, since, after the occlusion, the woman occupies the same space as the man did before the occlusion, confusing the appearance extractor of the model.

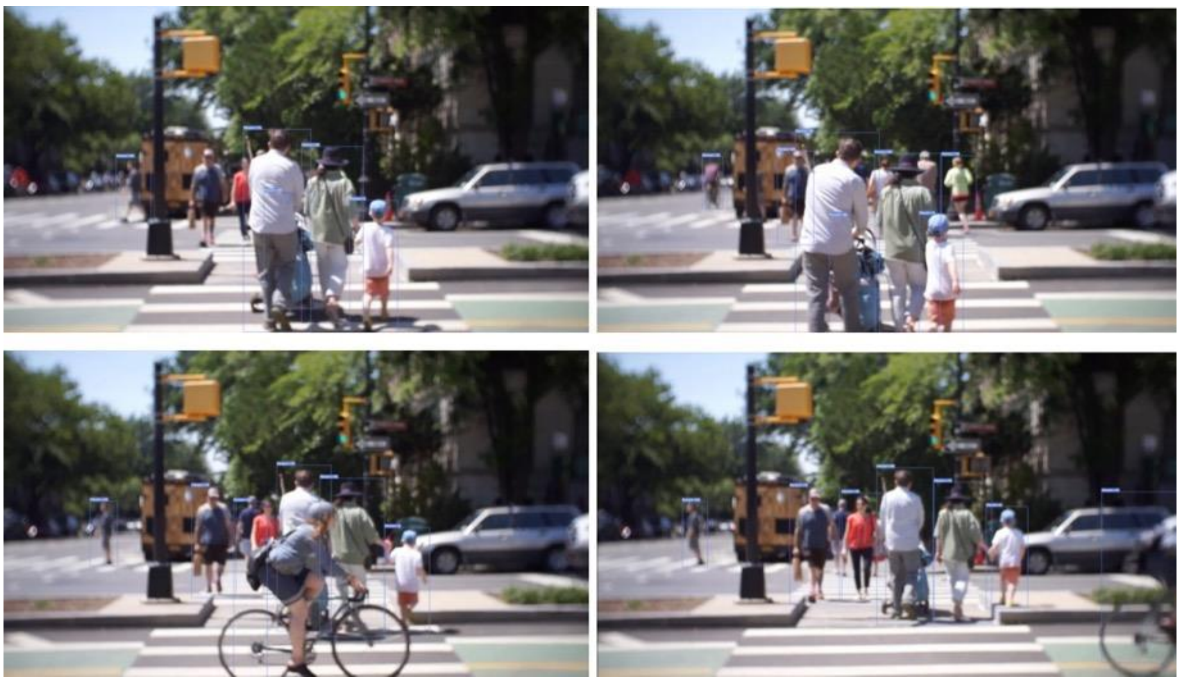


Figure 109. YOLOv3 + DeepSORT - Video 2

As said before, the YOLOv5 model that used from now on is the medium sized model trained on the CrowdHuman dataset. The results on the same videos are the following.

The results, as shown below, are very similar to the YOLOv3, at least at first glance. Taking a look at the videos shared in the **Annex II**, there is a slight improvement when an occlusion occurs, but nothing major. The main difference is the inference time, which in fact is a very important improvement.

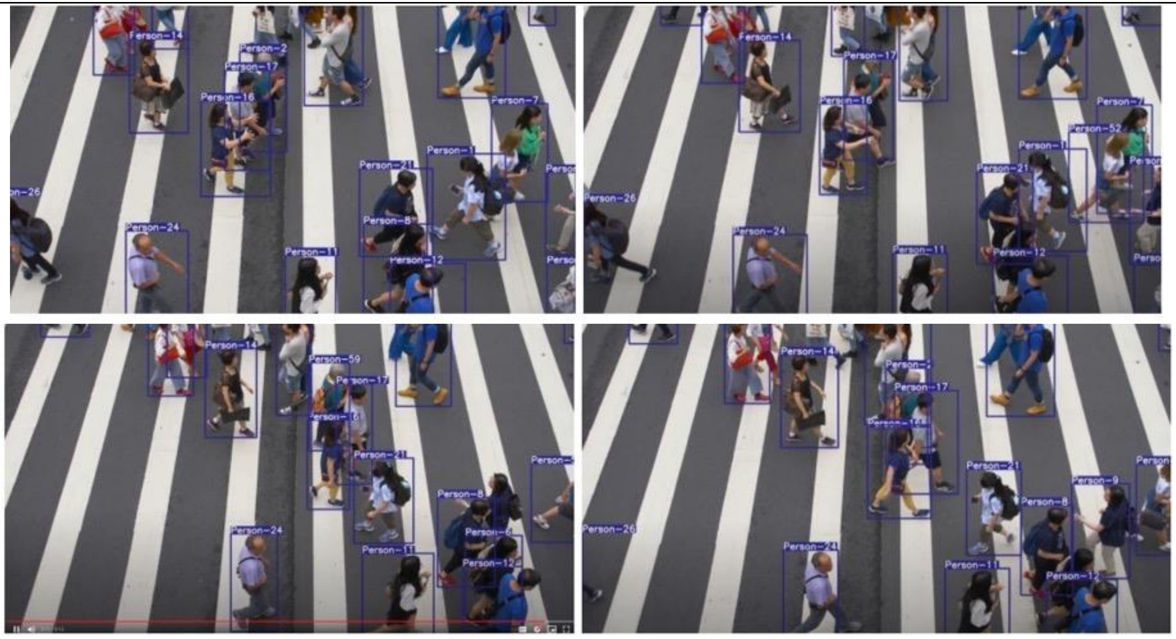


Figure 110. YOLOv5 + DeepSORT video 1

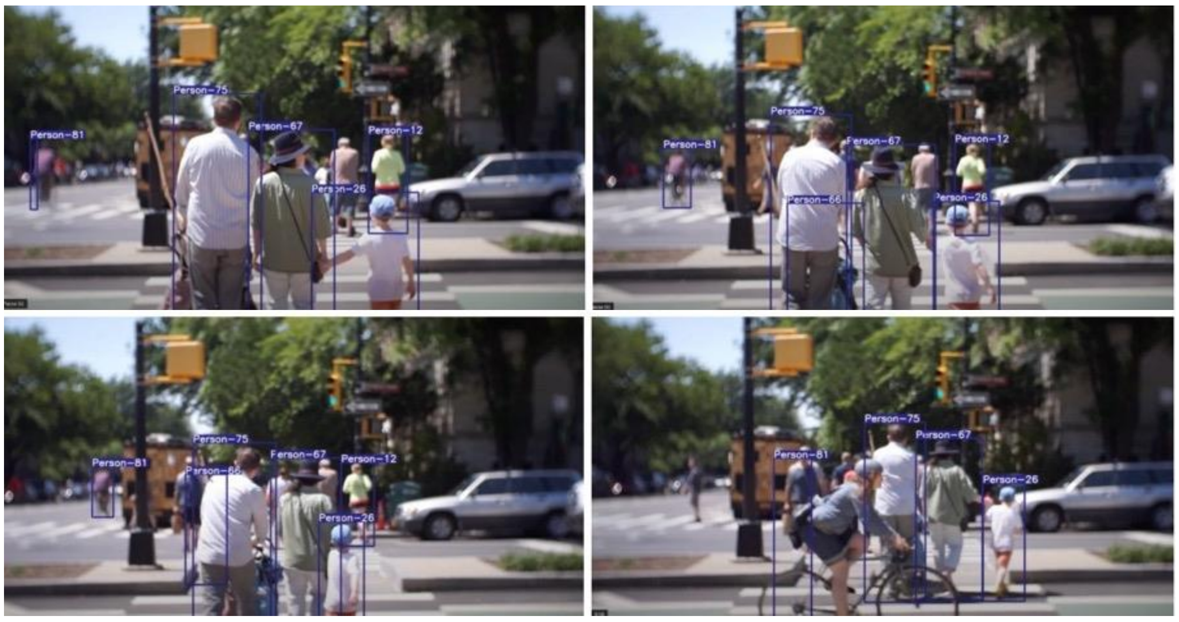


Figure 111. YOLOv5 + DeepSORT video 2

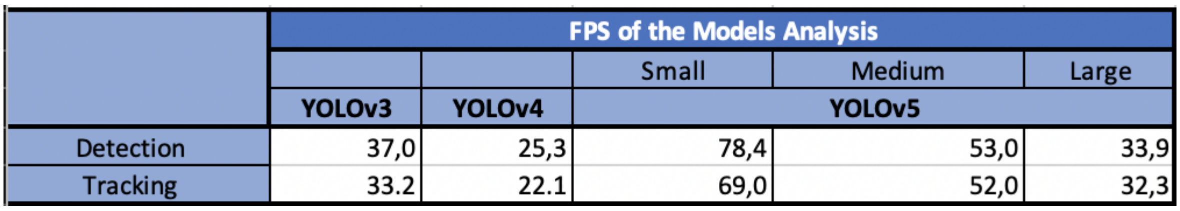
To be able to compare against the performance of other frameworks, apart from the DeepSORT model, only the SORT module was tried also. It does not make sense to provide the results in sequence images because the difference is not appreciated, but it can be found here.[48]

48 https://drive.google.com/drive/folders/1-gHliM6gyUGsbbFd-Ko0smRAdd42oNBp?usp=sharing

In the results provided, it is seen that the SORT algorithm has more trouble re-detecting lost detections between frames. Of course, this is normal and that is the whole point of DeepSORT, it provides a more accurate appearance descriptor to be able to be more accurate measuring similarities when trying to recover detections. In those results, because of the weaker similarity measure, it is also seen that the tracking is more unstable, that is because, since SORT loses more detections than DeepSORT, it gives the sense that the detections are jumping from positions, like it has some kind of lag. That is what makes DeepSORT more robust and stable and why it is the state-of-the-art algorithm for real-time tracking. In terms of FPS, SORT performs at 45 FPS when using the YOLOv5 model and 32 FPS for YOLOv3. Both perform in real-time, but they are a little bit slower than the DeepSORT version of both models.

6.3 INFERENCE TIME ANALYSIS OF THE RESULTS & CONCLUSIONS

As explained during this project, real-time processing is considered when the FPS measure is above 30 FPS. I only chose to show before the two models discussed before but, for the purpose of the study of the inference time, all models were tried to be able to compare them.



	FPS of the Models Analysis				
			Small	Medium	Large
	YOLOv3	YOLOv4	YOLOv5		
Detection	37,0	25,3	78,4	53,0	33,9
Tracking	33.2	22.1	69,0	52,0	32,3

Table 6: FPS analysis for the different models

It is shown that inference time is much better when using the YOLOv5 models. As shown, the smaller model is able to process more frames per second, but, if you remember the results of the detection stage, the smaller version is less accurate than the medium version. That is the tradeoff I had to decide to make. I consider that 52 FPS is sufficient enough to work in real time and the increased accuracy is always a good boost. Don´t forget that if this is put into production, the GPU is probably going to be more powerful than what Google Colab provides so the FPS will increase a little bit, making the system faster.

Looking at the other models, YOLOv4 is not only less accurate, but also it does not reach real-time processing capabilities. The larger YOLOv5 model barely reaches real-time, so despite it being a little bit more accurate, it is not considered to be quick enough. That is the advantage of implementing it natively instead of using the structure used in YOLOv3, which needs translation to Python, since it is originally built in C++.

The comparison with DeepSORT, as mentioned in the previous section, puts this new version ahead not only in robustness and stability but also in FPS, being DeepSORT 7 FPS quicker for YOLOv5 (the medium sized version) and 9 FPS faster in YOLOv3. All in all, applying deep convolutional networks makes the algorithm more robust and stable.

CHAPTER 7. DISTANCE ESTIMATION

For the last part of the project, the system needs to be able to estimate the distance between 2 individuals and be able to monitor it. There is nothing on the state of the art on this because this approach is not at all used in these tasks. There are a lot of examples of distance measuring, but they are almost all of them based on a parameter of distance to get depth and all that information hardcoded and fixed for each camera and perspective. The closest approach to a Deep Learning one is this one that transforms the camera view into bird´s view [54], that being a zenithal take of the same surroundings. The complication is that, before changing the perspective into a bird's view, you have to lock the objects you want to monitor [54]. That is not very practical at all for this project because with that, the system wouldn't be able to track new people entering the scene. You can see some examples of the bird's perspective changes below.

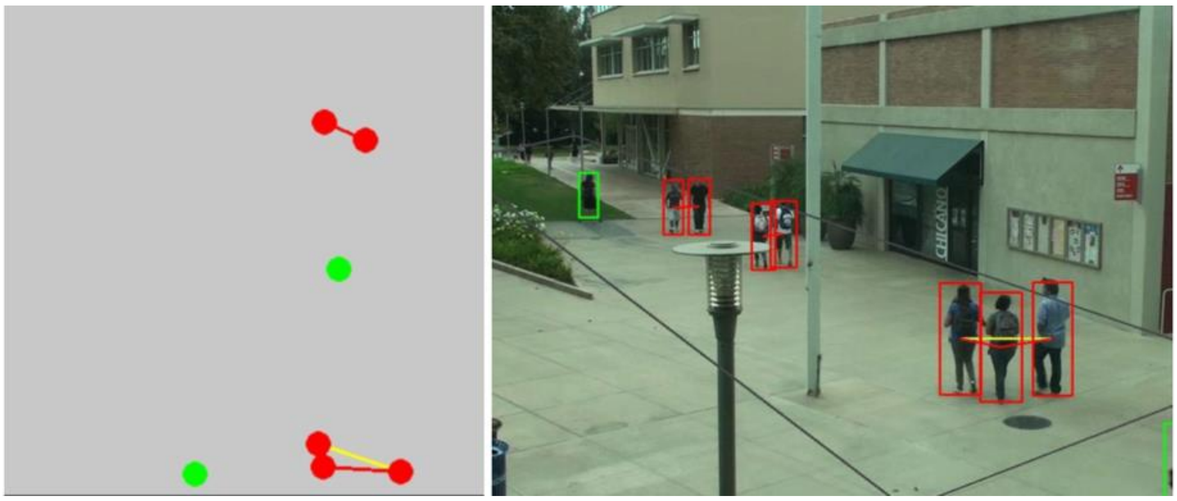


Figure 112. Example of Bird View Transformation [54]

As seen in the pictures and read in the post [54], to do this you need to establish a Region of Interest before starting the system. This is highly inefficient. As you can see in the picture, it is only able to monitor the distance to the regions of interest (the red points), but it is not able to monitor the distance if any of the green points go close to other points appearing in the scene. Of course, using the threshold method or the bird´s perspective transformation is very accurate. However, the objective is to try and build a system that is able to generalize and change from one shot to another and adapt and provide an estimation not based on thresholds or previously calculated regions of interest. Of course, this part of the project is highly experimental and there is no guarantee that the results obtained are perfect or even good enough.

The need for this last part is obvious. For some tasks, only with the tracking it may be enough to analyse and get the results wanted doing only tracking but, for an interpersonal distance monitoring, especially in this day and age, it is very important to provide this last stage and not only rely on our sight to identify the individuals that are too close together.

7.1 HOW IS IT GOING TO BE BUILT?

The first thing to do is to recognize what the model needs to be able to train for the estimation of distance. To clarify once again, the model to be built is some simple model with a couple convolutional layers and a lineal output, meaning that we see the problem as a regression problem. The model needs, first of all, a dataset of images. The previously used datasets are not valid because of many reasons. The first and most important one is that there are no annotations of the distance between the people in the images. Also, these are very complex datasets for this problem, at least at this stage where this task is relatively new. There are too many people, and it would be very difficult to estimate the distance between all the people. The goal is to simplify the problem and use simple images with only two images, and then see if the model is able to estimate the distance between only two people. The third thing that I need is some information of depth, which the datasets do not provide.

To do this I am going to generate a synthetic dataset using a simulator. With this I am going to be able to generate scenarios where there are two people and then measure the distance between them. These scenarios I am going to save them as images and the distances as the labels. This way I am going to construct my dataset. Using the simulator, I can also save the depth images, and use them as part of the dataset. Since I already have a detector that is very accurate, in order to help this final model, I am going to pass the images generated from the simulator, get the bounding boxes and put them on the depth images, this way, the model does not have to deal with colors or other information, it only needs to take the grayscale values of the depth and, with the help of the bounding boxes, estimate the distances provided in the labels. The following images explain this a little better.

The simulator I used is CoppeliaSim[49], which is a simulator built for robot tasks, but it is open source, and it might be enough.

[49] https://www.coppeliarobotics.com/

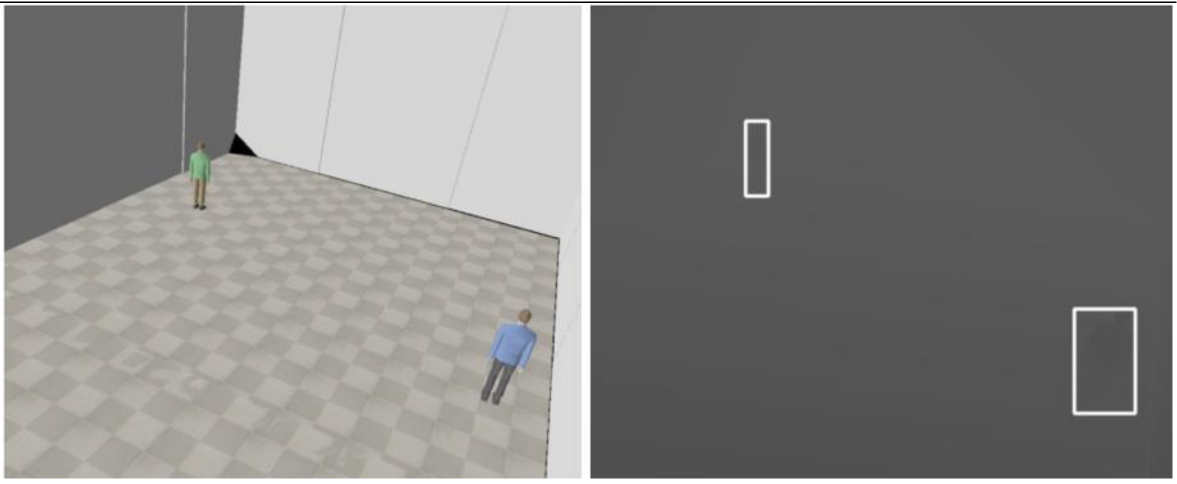


Figure 113. Original Images and Depth Images

The first image is the output of the simulator. As seen, it generates two people in random positions within the limits of the scenario. Those images are passed through the detector and the bounding boxes are “translated” to the depth image, as seen in the image on the right. You might see nothing on the depth images, that is ok, so do I. This is because the difference between pixel values is so very little and not perceivable to the human eye, but, in theory, the computer is able to differentiate them, let's hope it's true.

7.2 GENERATING THE DATASET

The first thing to do before trying any model is to generate the dataset I want to work with. As said before, none of the previously used dataset provides the distance between the labels annotated, and I could not find any dataset that has that.

To overcome this difficulty, I have used a simulator to generate persons on a 25 by 25 floor in random positions. The script I used to generate the different persons is provided below in different snippets. The first one is a function that takin a .ttm model, creates a person in the scenario created. This function is called twice, once for each individual in each iteration of the code.

```
def create_person(client_id: int, x: float, y: float, theta: float):
    current_path = os.path.dirname(os.path.abspath(__file__))
    model_path = os.path.join(current_path, 'persona.ttm')
    rc, out_ints, _, _, _ = sim.simxCallScriptFunction(client_id,
'ResizableFloor_5_25', sim.sim_scripttype_childscript, 'createRobot', [], [x, y,
theta], [model_path], "", sim.simx_opmode_blocking)
    person_handle = out_ints[0]

    return rc, person_handle
```
Snippet 1. Function to create persons

What this script basically does is connect to the remote API of CoppeliaSim, the simulator I am using and send the commands to the simulator's application. As seen in the code, each iteration of the code erases the people instances created in the previous ones, generates two new people in random positions, carefully bounded by the dimensions of the floor, measures the distance between them and saves, first, the image at that moment, the depth information at that same instant and the distance measured between the people. After all the iterations are completed, the script saves a CSV with all the distances, a NumPy vector with all the depth information and all the images. Another snippet of the code is presented below, it depicts how the individuals are generated, how the distance is measured and how it interacts with the vision sensor.

```
        ##Generating individuals in random positions
        position1 = (random.uniform(-4, 5), -0.75, random.uniform(-2, -11))
        position2 = (random.uniform(-4, 5), -0.75, random.uniform(-2, -11))
        try:
            sim.simxRemoveModel(client_id, person1_handle,
sim.simx_opmode_oneshot_wait)
            sim.simxRemoveModel(client_id, person2_handle,
sim.simx_opmode_oneshot_wait)
        except:
            print("Ya ha sido eliminado")

        # Create the persons
        try:
            _, person1_handle = create_person(client_id, position1[0],
position1[1], position1[2])
            _, person2_handle = create_person(client_id, position2[0],
position2[1], position2[2])
            sim.simxGetObjectPosition(client_id, person1_handle, -1,
sim.simx_opmode_streaming) # Initialize real position streaming
            sim.simxGetObjectPosition(client_id, person2_handle, -1,
sim.simx_opmode_streaming) # Initialize real position streaming
        except:
            print("No se ha generado la persona")

        # Execute a simulation step to get initial sensor readings
        sim.simxSynchronousTrigger(client_id)
        sim.simxGetPingTime(client_id)  # Make sure the simulation step has
finished

        start_time = time.time()

        # Distancia entre las personas
        p1 = np.array(position1)
        p2 = np.array(position2)
        distance_np = np.linalg.norm(p1-p2)
        distances.append(distance_np)
        print("\nDistance Between Individuals: {}".format(distance_np))
      res,res_image,image=sim.simxGetVisionSensorImage(client_id,v0,0,sim.simx_o
pmode_buffer)

        image_array = np.array(image, dtype=np.uint8)
        im = Image.frombuffer("RGB", (1024,1024), image_array, "raw", "RGB", 0,
1)

        ## DEPTH DATA
        rc,res_depth,depth_buffer = sim.simxGetVisionSensorDepthBuffer(client_id,
v0, sim.simx_opmode_buffer)
        depths.append(depth_buffer)
        depth_array = np.array(depth_buffer)
        depth = Image.frombuffer("L", (1024,1024), depth_array, "raw", "L", 0, 1)
```

Snippet 2. Creating the individuals, measuring and getting the information

After that, I needed to transform the depth information into actual images. For that I used a script that basically takes the NumPy input and separates it into the corresponding images. Finally, we put the images through the detector and move the bounding boxes to the depth images to help the model, and after it, we have the synthetic dataset. The first results are like in the image.

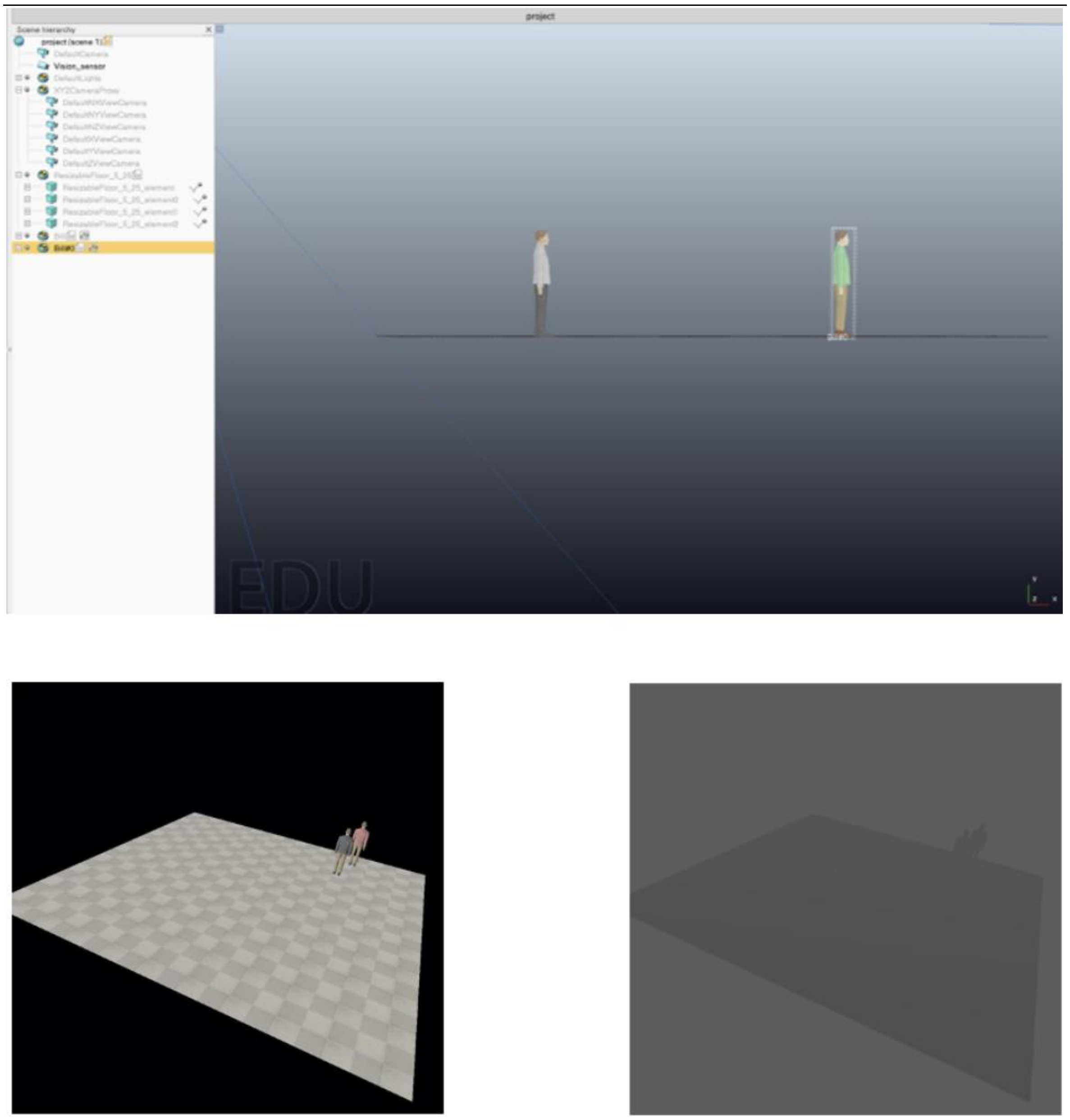


Figure 114. Simulator, Images and Depth Information

Of course, this did not come free of problems. The main problem is that, if not using walls in the simulator, as Figure 113 shows, the depth information that the simulator assigns is more or less random, and definitely not exact. It is true that you can identify somewhat clearly the individuals but only if some parts of them lie outside the floor boundaries. In theory this could not be a major problem but, when analyzing the pixel values inside the bounding boxes, which is, primarily what the model is going to consider, there is almost no difference between the pixel values corresponding to one person or another. For example, the closest person to the camera would have a pixel value of 80 and the farther person would have a pixel value of 84. Imagine if the individuals are so close to each other as in Figure 122. In this case the pixel value is the same for both (83), which will leave the model with only the position information as useful information to predict the distance between them.

To try to mitigate this, I put walls in the simulator scenario to try to "bound" the pixel values. While for the depth image, it might be more difficult to identify from a human point of view the people, by translating the bounding boxes, we know where they are and, now the range of pixel values is from 80 to 92. It is not the best possible (Ideally it would be 0-255) but it is 4 times better than before. This is one of the limitations of this simulator and having to transform manually the depth information to images, probably there is some lost information on that process, but the learning curve for the simulator is not easy and time was limited to figure how to adapt it to our problem. The final dataset looks something like in Figure 114:

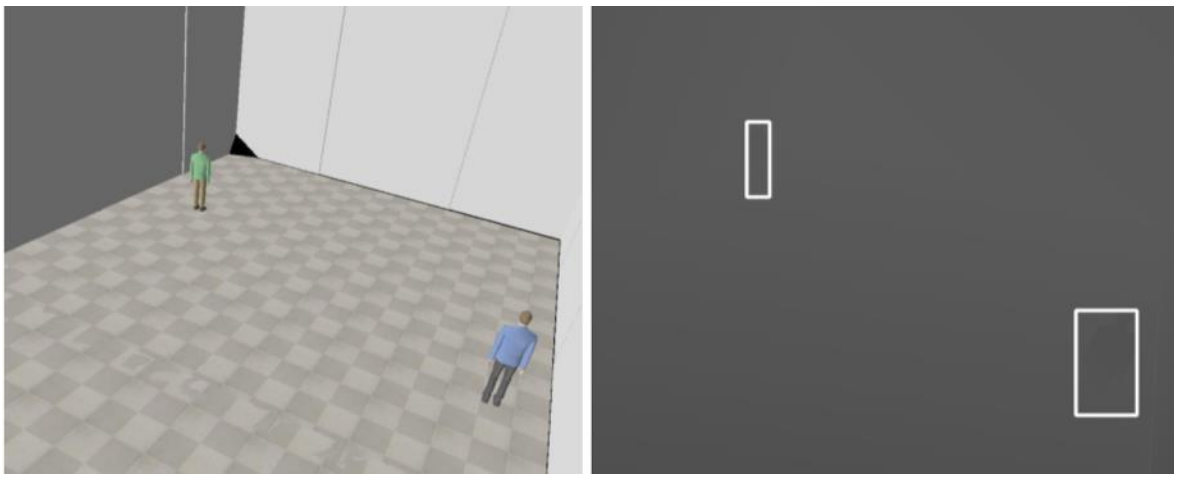
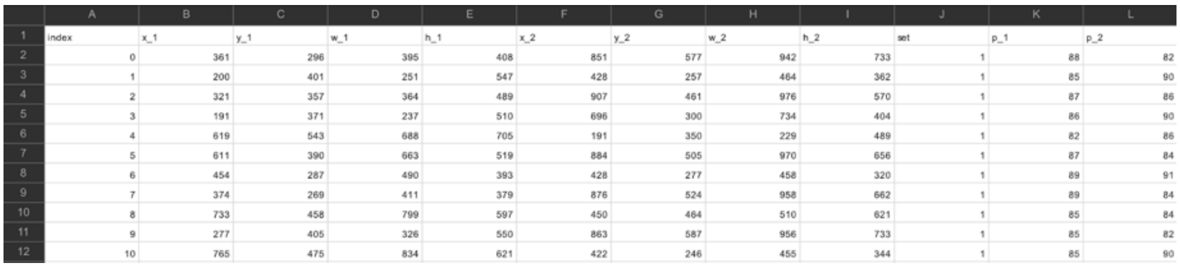


Figure 115. Final dataset structure

For each set of images, the corresponding distance is stored in a CSV file and matched later on the Python code.

To be able to try more things, a conventional CSV dataset was created with all the information to see if it is possible to develop the model on this information instead of iterating through to sets of images. The final result of this dataset is the as it is shown in the following sample:



	A	B	C	D	E	F	G	H	I	J	K	L
1	index	x_1	y_1	w_1	h_1	x_2	y_2	w_2	h_2	set	p_1	p_2
2	0	361	296	395	408	851	577	942	733	1	88	82
3	1	200	401	251	547	428	257	464	362	1	85	90
4	2	321	357	364	489	907	461	976	570	1	87	86
5	3	191	371	237	510	696	300	734	404	1	86	90
6	4	619	543	688	705	191	350	229	489	1	82	86
7	5	611	390	663	519	884	505	970	656	1	87	84
8	6	454	287	490	393	428	277	458	320	1	89	91
9	7	374	269	411	379	876	524	958	662	1	89	84
10	8	733	458	799	597	450	464	510	621	1	85	84
11	9	277	405	326	550	863	587	956	733	1	85	82
12	10	765	475	834	621	422	246	455	344	1	85	90

Figure 116. Full dataset in CSV format

As seen in this dataset the range of the pixel values is wider than with no walls, which definitely helped the models, as it will be shown later. The other columns are the center coordinates of the bounding boxes (x, y) and the height and width of the boxes (w,h), for both individuals. The other variable which is dropped when passing the rows to the models is the “set” one. I have constructed the dataset using 4 sets of images. The simulator can only generate people looking in one direction, so 4 subsets with people looking in the 4 main directions were generated: Front, backwards, to the left and to the right. This gives the dataset more variability and the ability to generalize better.

7.3 TRIAL AND ERROR

Since this is a very experimental phase, various experiments were done. TensorFlow was tried, since it is easier to build these simple models with, and try a transfer learning model and building another model from scratch. Finally, CSV dataframe is tried with a DNN model also. To do this, when passing the original images through the detector, the bounding boxes coordinates generated are going to be stored and after translating them to the depth images, the pixel value of the center of the bounding box can be computed and added to the dataframe, having some depth information to work with. The target column is still the distances generated by the simulator. This model will not be built with CNNs logically, but as a DNN with some dense layers and a linear output. In summary, I will provide 3 different models, TensorFlow + Transfer Learning, TensorFlow + New Model and the DNN model. But first, the results provided will be of training the model with the dataset generated with no walls in the scenario so you can see what is happening.

The dataset is read into the script by pairing the depth images to the corresponding distances.

The dataset is now ready to be fed into the model. Using a MobileNetV2 pre-trained model, the model is trained on this dataset by removing the last layer of the pretrained model and adding some dropout layers, some extra dense layers and a linear output, which is what needs to be used for a regression problem like this.

```
model_3 = MobileNetV2(weights=None, include_top=True, input_shape=(512, 512, 3))
x = model_3.get_layer(index=len(model_3.layers)-1).output

#Add a Dropout layer.
x = Dropout(0.3)(x)
x = Flatten()(x)
x = Dense(30, activation='relu',
kernel_regularizer=tf.keras.regularizers.L1(0.01),activity_regularizer=tf.keras.r
egularizers.L2(0.01))(x)
x = Dropout(0.3)(x)
x = Dense(15, activation='relu',
kernel_regularizer=tf.keras.regularizers.L1(0.01),activity_regularizer=tf.keras.r
egularizers.L2(0.01))(x)
x = Dropout(0.3)(x)
x = Dense(1, activation='linear')(x)
model_3 = Model(inputs = model_3.input,outputs = x)

model_3.summary()
```

Figure 117. Added layers to MobileNetV2 model

The training curves for this model and dataset are the following, with all the distances in meters:

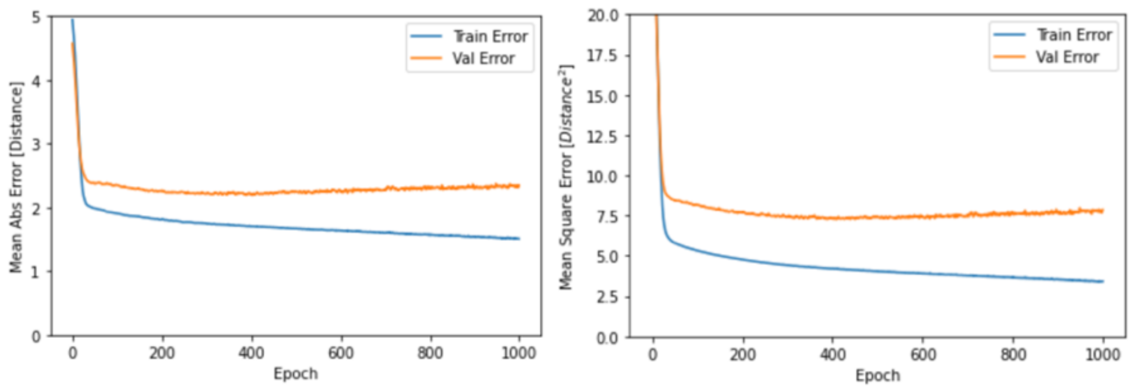


Figure 118. Training curves for the original dataset

As seen, the model is good in terms of stability but, it is not good in terms of loss, and clearly, it overfits even with dropout and regularization. For this problem, Mean Average Error measures the distance error in average of the model, and Mean Squared Error is the MAE squared. The MAE for this model on the training dataset is less than 2 meters, which means that, on average, the model misses by 2 meters to the target distance. I know that this is not an acceptable result, so that is why I decided to try and create a more precise dataset.

Let's now try the dataset generated with walls. The pairing is done exactly the same, but the result is obviously different, as seen in the following image.

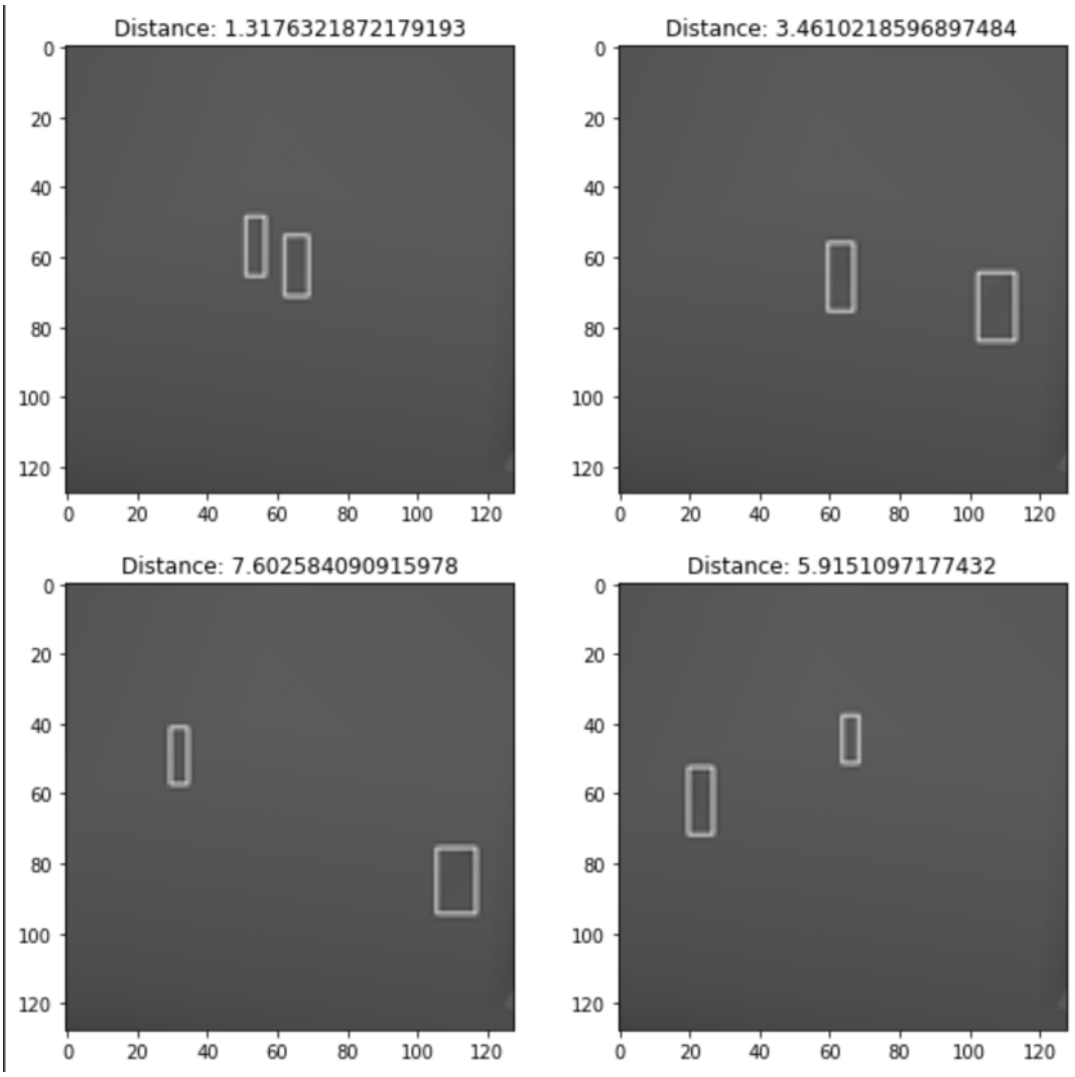


Figure 119. Pairing of Distances and Depth Images

As seen in the above image, the human eye cannot recognize the human shapes there, but hopefully with the pixel values and the bounding boxes, it is enough to estimate the distance. It is supposed that a computer is able to do it.

The first model is the exact same model as before, the only difference is that the input is this new dataset, whose objective is to compare and see if there is any improvement from the other dataset.

As seen in the following curves, the model is less stable but much more precise, which for this problem, I think is a much better option because it is going to be able to differentiate better between pixels. MAE for this model is 0.93 meters in the training dataset and 1.2 in the test dataset, which is considerably better than the previous model, and it is visually proven in the scatter plot depicted in figure 120. It also does not overfit as the previous model did.

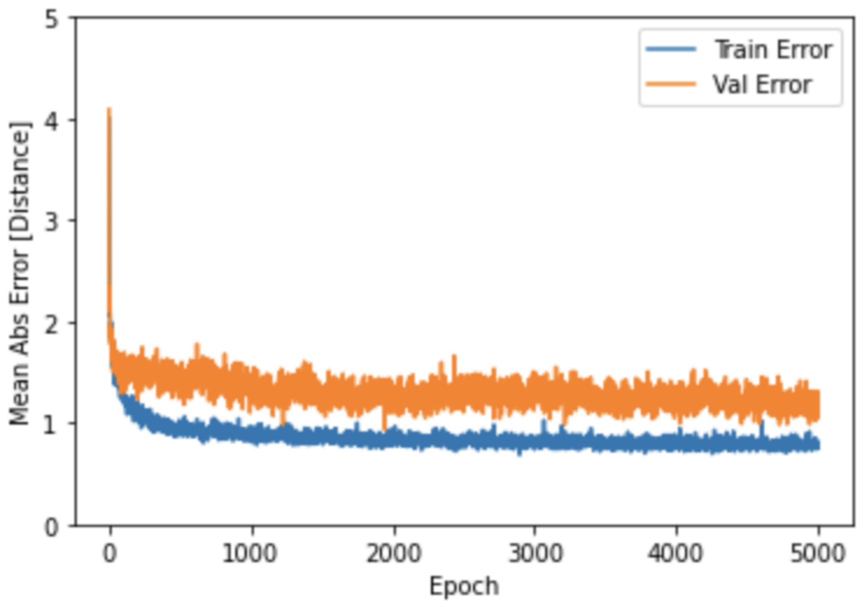


Figure 120. Training curves for Transfer Learning model on new dataset

If we put in a scatter plot the predicted outputs against the supposed results, we get the following result. It is not the best, but it is an acceptable result for now. The predicted outputs follow more or less the diagonal between true values and predictions. It is true that this model has some trouble predicting short distances, which is kind of the objective but, for now this is the best model. Let's keep in mind that a lot of room for improvement can be done in creating a much better dataset and using a much more powerful simulation tool, but this is the resources we have to try and figure out if predicting distance is possible.

The architecture of the CNN model is as follows. Using a pretrained model like MobileNetv2[50], the model used adds dropout and a flattening layer to the output of the pretrained model. It adds after it a 30-neuron fully connected layer and a 15-neuron fully connected after ir. With a dropout layer afterwards, the models' output is a single neuron with a linear activation function. All distances in the scatter plot are in meters.

[50] K. Dong, C. Zhou, Y. Ruan and Y. Li, "MobileNetV2 Model for Image Classification," 2020 2nd International Conference on Information Technology and Computer Application (ITCA), 2020, pp. 476-480, doi: 10.1109/ITCA52113.2020.00106.

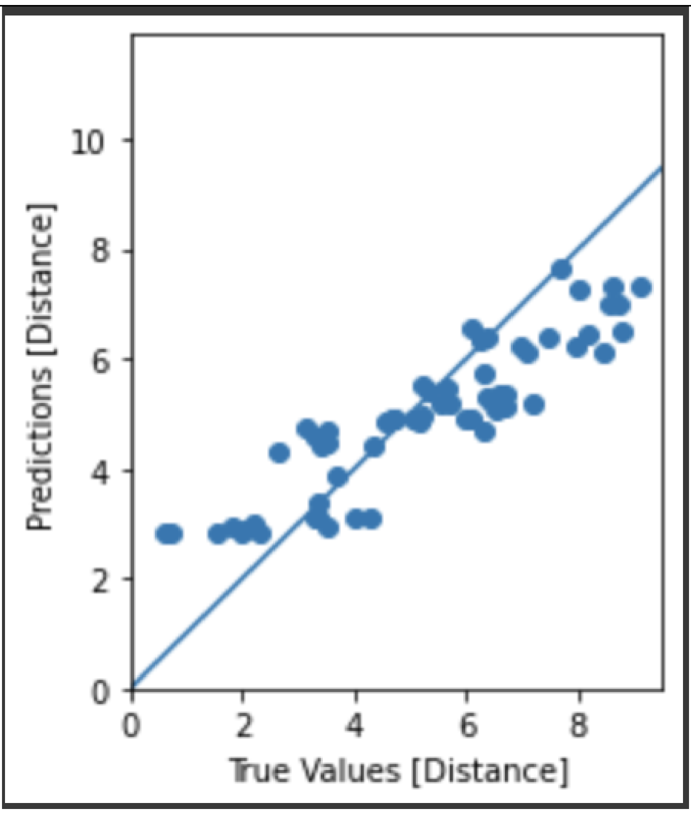


Figure 121. Scatter plot for Transfer Learning Model

Now, after demonstrating that this new dataset is a much better base to work with, I am going to try a much simpler model built from scratch and see if the same results (Or better) can be reached with much less computational power. The architecture for the model is very simple, it is built with 4 convolutional layers with dropout and batch normalization followed by 3 dense layers with a linear output. I´ve added regularization to the convolutional layers to try and increase stability. The code for the architecture and it's hyperparameters is as depicted in the next figure.

```
model_4 = Sequential()
# padding same works
model_4.add(Conv2D(10, 41, padding='same', input_shape=(256, 256, 3),
activation='relu'))
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D((2,2)))
model_4.add(Dropout(0.4))
model_4.add(Conv2D(8, 10 ,padding='same', activation='relu'))
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D((2,2)))
model_4.add(Dropout(0.3))
model_4.add(Conv2D(16, 4 ,padding='same', activation='relu'))
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D((2,2)))
model_4.add(Dropout(0.3))
model_4.add(Conv2D(8, 4, padding = 'same', activation='relu'))
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D((2,2)))
model_4.add(Dropout(0.3))
model_4.add(Flatten())
model_4.add(Dense(30,
activation='relu',kernel_regularizer=tf.keras.regularizers.L1(0.01),activity_regu
larizer=tf.keras.regularizers.L2(0.001)))
model_4.add(BatchNormalization())
model_4.add(Dropout(0.3))
model_4.add(Dense(20, activation='relu',
kernel_regularizer=tf.keras.regularizers.L1(0.01),activity_regularizer=tf.keras.r
egularizers.L2(0.001)))
model_4.add(Dense(5, activation='relu'))
model_4.add(Dense(1, activation='linear'))

sgd = SGD(lr=0.0001, decay=1e-5, momentum=0.9, nesterov=True)

model_4.compile(loss='mean_squared_error',optimizer=Adam(lr=0.0001),
metrics=['mae'])
```

Figure 122. Code for the CNN model built from scratch

As seen in the following chart, this model can reach, sometimes, a better accuracy than before but it is much less stable than before. This is very logical since it is a much simpler model compared to the pretrained one that has hundreds of layers, not only 10-12 like this model has. All distance are in meters.

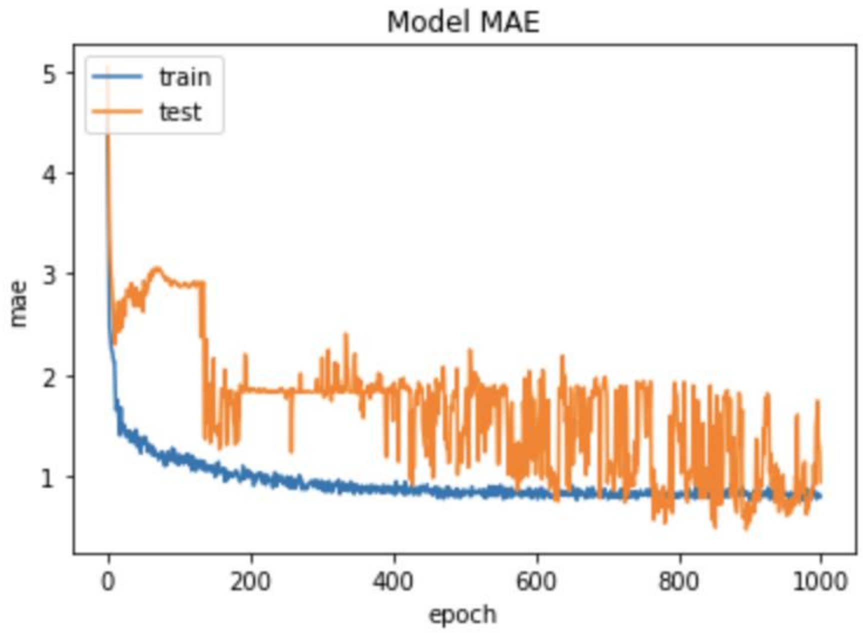


Figure 123. Training curve for Model from scratch

Plotting the scatter plot (in meters) for the model, we see that this model is more biased than the previous model. It is also much more unstable. This makes the MAE be higher on average and very variable, since for one batch, it can be 0.8 meters and for the next it could be 1.7 meters. It looks like this model can deal better with short distances, but this is not a fact that I would consider to be set in stone since that could mean that for this batch plotted, I got lucky and got the parameters that provide a 0.8 meters MAE and, also, it does not deal that much better with short distances.

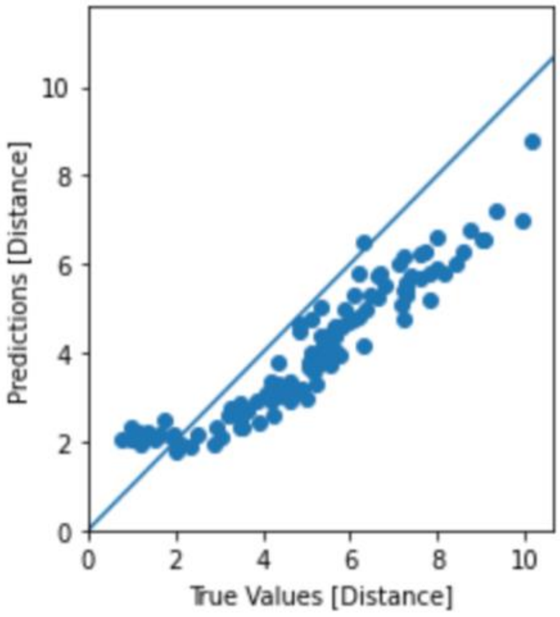


Figure 124. Scatter plot for the model built from scratch

Finally, I am going to try a model based on DNNs with the CSV dataset I showed earlier. The architecture of this model is very simple, and the code is implemented as follows:

```
def build_model():
  model_df = keras.Sequential([
    layers.Dense(32, input_shape=[1,len(train_dataset.keys())],
kernel_regularizer=tf.keras.regularizers.L1(0.01),activity_regularizer=tf.keras.r
egularizers.L2(0.01)),
    layers.LeakyReLU(),#, input_shape=[1,len(train_dataset.keys())]),
    layers.Dropout(0.5),
    layers.Dense(64,
kernel_regularizer=tf.keras.regularizers.L1(0.001),activity_regularizer=tf.keras.
regularizers.L2(0.001)),
    layers.LeakyReLU(),#, input_shape=[1,len(train_dataset.keys())]),
    layers.Dropout(0.3),
    layers.Dense(64,
kernel_regularizer=tf.keras.regularizers.L1(0.001),activity_regularizer=tf.keras.
regularizers.L2(0.001)),
    layers.LeakyReLU(),
    layers.Dropout(0.5),
    layers.Dense(32,
kernel_regularizer=tf.keras.regularizers.L1(0.001),activity_regularizer=tf.keras.
regularizers.L2(0.001)),
    layers.LeakyReLU(),

layers.Dense(16,kernel_regularizer=tf.keras.regularizers.L1(0.001),activity_regul
arizer=tf.keras.regularizers.L2(0.001)),
    layers.LeakyReLU(),
    layers.Dropout(0.3),
    layers.Dense(1, activation = 'linear')
  ])

  optimizer = tf.keras.optimizers.Adam(0.001)
  model_df.compile(loss='mean_squared_error', optimizer=optimizer,metrics=['mae',
'mse'])
  return model_df
```

Figure 125. Code for the DNN model

Even though this is a different approach, the structure is pretty similar. The dense layers substitute the CNNs and the output is linear once again. Training the model for 1000 epochs, the training curves are as follows:

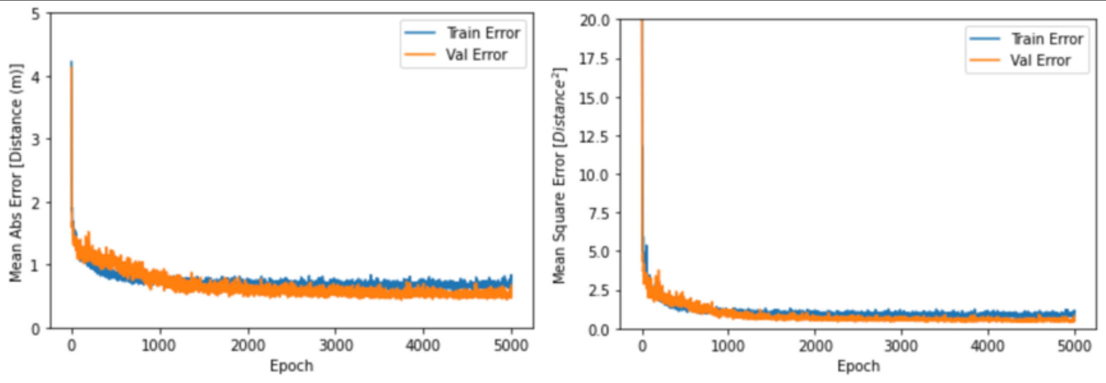


Figure 126. Training curves for DNNs model

Even though this model is affected by noise, it does not overfit and is more stable. It still shows a little bias, but less than the previous model. It also has much less variance than the first CNN model, the one with the pre-trained model. In summary, it reduces the bias from the model built from scratch and reduces the variance from the model with transfer learning, meaning that is better than both of them. The only drawback it has is that it still suffers a little bit when trying to predict either very long distance or very short ones, which is the real problem, although it deals better with short ones than long ones. These improvements, translated to a metric mean that the MAE of this model is 0.6 meters, that, while not perfect, it is much less and more stable than previous models.

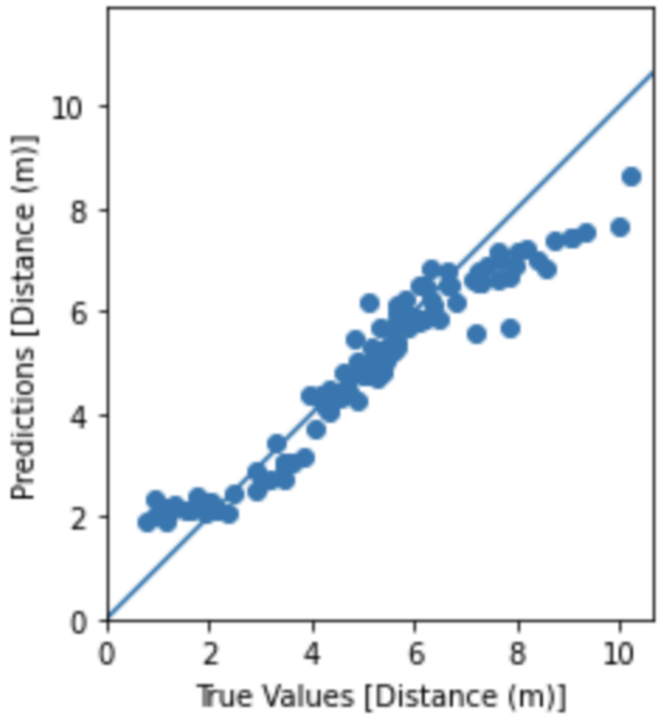


Figure 127. Scatter plot for DNNs model

7.4 CONCLUSIONS

So, in conclusion, using a DNN model is much more precise and stable than using a CNN model. The por performance of the CNN model is probably caused by a dataset that is not very big (600 images), and a simulator that either is used by an expert on it or is not good enough. A better simulator could provide better depth information and therefore help the models to be much better. This last reason applies to the DNN model also, because of the use of the pixel values from the depth information as features. Another reason is that in order to not run out of memory with the GPU, the resolution of the input images had to be reduced, difficulting the extraction of features of the model (From 1024 pixels to 256). With a more powerful GPU or better use of the memory, the resolution of the input images could be increased and give the model more and better information to train on.

But not only the CNN results could be improved by these measures, also the DNN model would be greatly improved with a better dataset. If the pixel values have a wider range, the DNN would be much more precise when estimating the distance and, of course, if the dataset has 6000 images (rows in this case) instead of 600, the variability would be higher, and the estimation could be way more precise. But my computer cannot generate 6000 thousand images and depths without collapsing since each set of 600 images and depths weights 20GB.

In conclusion, it is possible to estimate the distance between individuals using a deep learning approach with two different methods. Using a CNN or a DNN, distance can be estimated. Of course, all these models and datasets can be very much improved, but it serves me well to demonstrate that, for the task presented, monitoring the interpersonal distance using a Deep Learning approach is possible.

CHAPTER 8. CONCLUSIONS & FUTURE WORK

Considering the results presented, mainly in the last section, I believe that the monitorization of the interpersonal distance using Deep Learning while having a processing speed within the boundaries of real time presented throughout the project (30 FPS) is possible. However, I´ll provide now a brief summary of the results of each section and the conclusions that can be drawn from them.

For the detection stage, 9 different detectors spread out in 4 different datasets (COCO, KITTI, Caltech and CrowdHuman) and 3 different models have been tried. Trying them in chronological order, first YOLOv3, then YOLOv4 and finally YOLOv5, I was able to discern why each of them perform the way they do. Starting with YOLOv3, it provided a relatively low accuracy for the COCO dataset but a very reasonable one for both KITTI and CrowdHuman. However, in inference, the COCO model was the second best performing one just behind CrowdHuman. KITTI was not able to generalize that well due to the nature of its dataset, as explained in the corresponding section. In terms of FPS, this model provided a speed of around 30 FPS, which is very much on the limit of real time.

The evolution of YOLOv3 into YOLOv4 should have meant better results, but that was not the case. YOLOv4 model is a much deeper model that, in training, it's true that the results were better than YOLOv3, but in inference, that difference in accuracy was not tangible. In fact, there was only a slight improvement in the COCO dataset, but not in CrowdHuman or KITTI, the worst performing one of them all clearly. In terms of FPS, since this is a much heavier model than YOLOv3, its performance was far from real time processing, so I decided to discard all YOLOv4 models for the next stages, they were less accurate and slower than YOLOv3. Because of the incapacity of generalizing, KITTI dataset was also discarded for new stages, CrowdHuman and COCO were simply better.

Then YOLOv5 appeared in the scene and the results were all better than what we saw before. Not only in training, that the smaller model was already more accurate than all the previous models, but also on inference time. As explained before, YOLOv5 has the same structure and advances as YOLOv4, but it is much faster and precise because it is built natively on PyTorch, not in C++ like YOLOv4 and YOLOv3 model is. Reducing all the translating and parsing of documents and a simpler structure was enough to have much more accuracy and reduced the inference time considerably, reaching a 60 FPS processing speed for the model selected, double the value we established as the limit. In terms of datasets, CrowdHuman was the best performing one by a considerable distance, so I decided to stick with the YOLOv5 model trained on CrowdHuman dataset for the next stage.

So far, we have a very accurate model, that has some difficulties detecting people from very long distances and has some trouble with occlusions. However, it is very fast. For now, long distance detection is not a problem of much concern, as long as short distance detection is good, which it is. The occlusions, however, are a problem and there´s need to deal with them. That is why we incorporate a tracking mechanism. It´s objective is to have some sense of memory and not loose detections when occlusions happen. To do this the method used was the state-of-the-art algorithm for real-time tracking, which is DeepSORT. It is based on Kalman filters but incorporates a deep convolutional network that works as an appearance descriptor between frames. Comparing the output of it to determine if a detection is very similar to one in the previous frame, we can "recover" lost detections some frames later. That process has been explained in detail in the appropriate section and it is key to the objectives of the project.

Looking at the results and inference time of the YOLOv5 model + DeepSORT model, the outcome is very satisfactory. Occlusions are corrected in a lot of cases. Of course, this is not an exact science and between frames, as explained, one detection can occupy the space of another and be very similar. Unfortunately, some of those occlusions are not corrected, but that is rare and overall performance is very good. In terms of FPS, the processing speed is obviously reduced since the images pass through another model, but the FPS of the models selected are still above 50 FPS, a speed well above the limits stablished. Since the last stage is a very simple and light model, the processing speed is not expected to be reduced so, the real-time objective is reached with a considerable margin.

For the last stage, there was a problem that when designing the project, we did not expect. The datasets we used are not usable. To estimate the distance, we need some kind of annotations to train the model against them and check accuracy etc. There are no datasets like this so, in order to reach a deep learning solution for the problem, a synthetic dataset was created. For this we used a simulator and generated a dataset of 600 images of 2 individuals with the distance between them measured and stored as the labels. This is very archaic, and it is not a state-of-the-art dataset but, as shown in the results, is good enough for now. While using a very simple dataset and a very simple DNN model, we reached an MAE of only 0.6 m, that is not excellent but is good enough. In the future work section it will be discussed how that can be improved.

With all these stages producing good results, estimating the distance between individuals is possible using a deep learning approach and with real time processing speed. Of course, there is much work ahead, but this could be a first step towards a much more general and precise solution in the future and, since all this was completely new to me 8 months ago, it is a very good first step in that direction.

FUTURE WORK

In terms of detection and tracking, there is much to improve, but since the FPS is very good in the project, which was the main objective of these stages, it is an area more researched, and the part that needs more improvements is distance estimation, I´ll focus on what can we do to improve that last one.

To begin with, generating a better dataset is key. As mentioned, the simulator we used, is not designed for this type of tasks. The right thing to do is to use a photorealistic simulator based on Unity[51], for example and gather and build the dataset from it. Not only that, but I had to learn in a very short amount of time how to use it, and even though we managed to generate a good scenario and record the distance, I cannot say that I am an expert on the simulator. Maybe with more knowledge about it we could generate better images and depth information. As mentioned in the previous section, pixels values range ws very small and, therefore, not very precise since, ideally, a range between 0 and 255 is what the simulator can give. Better understanding of the simulator would provide a much more precise dataset but, also, a bigger dataset is needed. I was able to generate 600 images and create the dataset but having more time to generate a bigger dataset could be of great help. Time was limited and what I got to generate was good enough to complete the objective of the project.

Apart from that, which I believe would be enough to provide a very good solution, a deeper DNN model with a more complex structure could be of help also. Since we have more than 20 FPS of margin between the actual system with the selected model and the real time limit of 30 FPS (with the GPUs provided by Google Colab, if the GPU is more powerful, the FPS would obviously improve). Since the system is able to work well above 30 FPS, investing in a deeper model should improve the results accordingly, even if it slows the process a little bit.

As an end note, incorporating that last stage into the pipeline could be done more efficiently since, now, the input depth information is reworked into the 80-92 range of pixel values to be correctly inferenced. With a better dataset and better simulator knowledge, we could train the DNN with a 0-255 range and then, no reworking of the input depth information would be needed, but, for now, that is the way it is done.

[51] https://unity3d.com/es/real-time-rendering-3d

CHAPTER 9. BIBLIOGRAPHY

[1] M. Cote and A. B. Albu, "Teaching Computer Vision and Its Societal Effects: A Look at Privacy and Security Issues from the Students' Perspective," 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI, 2017, pp. 1378-1386, doi: 10.1109/CVPRW.2017.180.

[2] M. Lauronen, "Ethical Issues In Topical Computer Vision Applications", 2017, University of Jyväskylä.

[3] Computer Vision Tools: https://www.analyticsinsight.net/top-10-computer-vision-tools-for-2020/

[4] Tensorflow Documentaton: https://www.tensorflow.org

[5] Google Research & Resources: [52] [53] [54] [55]

[6] K. Sage and S. Young, "Security applications of computer vision," in IEEE Aerospace and Electronic Systems Magazine, vol. 14, no. 4, pp. 19-29, April 1999, doi: 10.1109/62.756080.

[7] G. Rigoll, B. Winterstein and S. Muller, "Robust person tracking in real scenarios with non-stationary background using a statistical computer vision approach," Proceedings Second IEEE Workshop on Visual Surveillance (VS'99) (Cat. No.98-89223), Fort Collins, CO, USA, 1999, pp. 41-47, doi: 10.1109/VS.1999.780267.

[8] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale and S. Shafer, "Multi-camera multi-person tracking for EasyLiving," Proceedings Third IEEE International Workshop on Visual Surveillance, Dublin, Ireland, 2000, pp. 3-10, doi: 10.1109/VS.2000.856852.

[9] J. C. Silveira Jacques Junior, S. R. Musse and C. R. Jung, "Crowd Analysis Using Computer Vision Techniques," in IEEE Signal Processing Magazine, vol. 27, no. 5, pp. 66-77, Sept. 2010, doi: 10.1109/MSP.2010.937394.

[10] A. Krogh, "What are artificial networks", 2008, Department of Biology and Biotech Research and Innovation Centre, University of Copenhagen

[11] A. B. Chan and N. Vasconcelos, "Bayesian Poisson regression for crowd counting," 2009 IEEE 12th International Conference on Computer Vision, Kyoto, 2009, pp. 545-551, doi: 10.1109/ICCV.2009.5459191.

[52] https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43022.pdf
[53] https://research.google/teams/perception/
[54] https://ai.googleblog.com/2017/06/supercharge-your-computer-vision-models.html
[55] https://store.google.com/us/product/nest_aware?hl=en-US&GoogleNest

[12] B. Leibe, E. Seemann and B. Schiele, "Pedestrian detection in crowded scenes," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 878-885 vol. 1, doi: 10.1109/CVPR.2005.272.

[13] A. N. Marana, L. F. Costa, R. A. Lotufo and S. A. Velastin, "On the efficacy of texture analysis for crowd monitoring," Proceedings SIBGRAPI'98. International Symposium on Computer Graphics, Image Processing, and Vision (Cat. No.98EX237), Rio de Janeiro, Brazil, 1998, pp. 354-361, doi: 10.1109/SIBGRA.1998.722773.

[14] X. Wu, G. Liang, K. K. Lee and Y. Xu, "Crowd Density Estimation Using Texture Analysis and Learning," 2006 IEEE International Conference on Robotics and Biomimetics, Kunming, 2006, pp. 214-219, doi: 10.1109/ROBIO.2006.340379.

[15] Joel Janai; Fatma Güney; Aseem Behl; Andreas Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art," in Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art , now, 2020.

[16] N. Deepika and V. V. Sajith Variyar, "Obstacle classification and detection for vision based navigation for autonomous driving," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 2092-2097, doi: 10.1109/ICACCI.2017.8126154.

[17] SegNet Project Page: https://mi.eng.cam.ac.uk/projects/segnet/

[18] Patel, Rachana & Patel, Sanskruti. (2020). A Comprehensive Study of Applying Convolutional Neural Network for Computer Vision. International Journal of Advanced Science and Technology. 6. 2161-2174.

[19] W. Luo, A. G. Schwing and R. Urtasun, "Efficient Deep Learning for Stereo Matching," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 5695-5703, doi: 10.1109/CVPR.2016.614.

[20] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[21] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, "Deep Learning for Computer Vision: A Brief Review", Computational Intelligence and Neuroscience, vol. 2018, Article ID 7068349, 13 pages, 2018. https://doi.org/10.1155/2018/7068349

[22] W. Lan, J. Dang, Y. Wang and S. Wang, "Pedestrian Detection Based on YOLO Network Model," 2018 IEEE International Conference on Mechatronics and Automation (ICMA), Changchun, 2018, pp. 1547-1551, doi: 10.1109/ICMA.2018.8484698.

[23] M. Javad Shafiee, B. Chywl, F. Li, A. Wong, “Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video”, 2017

[24] Varior R.R., Shuai B., Lu J., Xu D., Wang G. (2016) “A Siamese Long Short-Term Memory Architecture for Human Re-identification”. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9911. Springer, Cham. https://doi.org/10.1007/978-3-319-46478-7_9

[25] R. Rama Varior, G. Wang, M. Haloi, “Gated Siamese Convolutional Neural Network Architecture for Human Re-Identification”, 2016

[26] A. Milan, S. Hamid Rezatofhighi, A. Dick, I. Reid, K. Schindler, “Online Multi-Target Tracking Using Recurrent Neural Networks”, 2016

[27] J. Ju, D. Kim, B. Ku, D. K. Han and H. Ko, "Online multi-person tracking with two-stage data association and online appearance model learning," in IET Computer Vision, vol. 11, no. 1, pp. 87-95, 2 2017, doi: 10.1049/iet-cvi.2016.0068.

[28] Ming-xin Jiang, Chao Deng, Zhi-geng Pan, Lan-fang Wang, Xing Sun, "Multiobject Tracking in Videos Based on LSTM and Deep Reinforcement Learning", Complexity, vol. 2018, Article
ID 4695890, 12pages, 2018. https://doi.org/10.1155/2018/4695890

[29] B Jiang, P Zhang, L Huang, “Visual Object Tracking by Segmentation with Graph Convolutional Network”

[30] S. Pellegrini, A. Ess, K. Schindler and L. van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," 2009 IEEE 12th International Conference on Computer Vision, Kyoto, 2009, pp. 261-268, doi: 10.1109/ICCV.2009.5459260.

[31] NVidia Research in CV: https://www.nvidia.com/en-us/research/computer-vision/

[32] J. Gu, X. Yang, S. De Mello and J. Kautz, "Dynamic Facial Analysis: From Bayesian Filtering to Recurrent Neural Network," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 1531-1540, doi: 10.1109/CVPR.2017.167.

[33] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, Shuai Zheng, Yi Zhu, “GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing”, 2020

[34] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.

[35] K. He, X. Zhang, S. Ren and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904-1916, 1 Sept. 2015, doi: 10.1109/TPAMI.2015.2389824.

[36] R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.

[37] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[38] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[39] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement" 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1804.02767.

[40] Detectron2 - A Facebook Research for Computer Vision and Object Detection https://paperswithcode.com/lib/detectron2

[41] Wildlife Monitoring with CV - https://louis030195.medium.com/cloud-computer-vision-for-wildlife-monitoring-a50bfee6bef

[42] Medicine and CV - https://www.exxactcorp.com/Whitepapers/Discover-AI-in-Medical-Imaging

[43] A. Bochkowsky, C. Y Wang and H.Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, 2020 ArXiv, 2004.10934

[44] C. Wang, H. Mark Liao, Y. Wu, P. Chen, J. Hsieh and I. Yeh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2020, pp. 1571-1580, doi: 10.1109/CVPRW50498.2020.00203.

[45] D. Salvi, J. Waggoner, A. Temlyakov and S. Wang, "A graph-based algorithm for multi-target tracking with occlusion," 2013 IEEE Workshop on Applications of Computer Vision (WACV), 2013, pp. 489-496, doi: 10.1109/WACV.2013.6475059.

[46] Nanonets: DeepSORT - https://nanonets.com/blog/object-tracking-deepsort/

[47] Meanshift Algorithm - http://www.chioka.in/meanshift-algorithm-for-the-rest-of-us-python/

[48] Kalman Filters - Object Tracking: Kalman Filter with Ease - CodeProject

[49] Deep Recurrent Networks - https://davheld.github.io/GOTURN/GOTURN.html

[50] G. Ning et al., "Spatially supervised recurrent convolutional neural networks for visual object tracking," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050867.

[51] N. Wojke, A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.

[52] MOT (Multiple Object Tracking) Challenge - MOT Challenge

[53] Deepak Biria: https://blog.usejournal.com/social-distancing-ai-using-python-deep-learning-c26b20c9aa4c

[54] Lowe, D.G. Distinctive Image Features from Scale-InvariantKeypoints. *International Journal of Computer Vision* **60,** 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

ANNEX I: DATA LOADING AND FORMATTING SCRIPTS

COCO_PERSONS.PY

Script that downloads, making use of COCO API, only images that contain annotations of people.

```
from PythonAPI.pycocotools.coco import COCO
import numpy as np
import requests
import matplotlib.pyplot as plt
import pylab
import argparse
import os
pylab.rcParams['figure.figsize'] = (8.0, 10.0)

if __name__ == "__main__":
   parser = argparse.ArgumentParser()
   parser.add_argument("--options", type=int, default=0, help="0 for training
dataset, 1 for validation dataset")
   opt = parser.parse_args()
   print(opt)

   if opt.options == 0:
       dataType='train2014'
   elif opt.options == 1:
       dataType='val2014'
   else:
       print("Error")

   annFile='annotations/instances_{}.json'.format(dataType)

   coco = COCO(annFile)
   cats = coco.loadCats(coco.getCatIds())
   nms=[cat['name'] for cat in cats]
   print('COCO categories: \n{}\n'.format(' '.join(nms)))

   catIds = coco.getCatIds(catNms=['person']) ## TO KEEP ONLY PERSON IMAGES
   imgIds = coco.getImgIds(catIds=catIds)
   images = coco.loadImgs(imgIds)

   images = coco.loadImgs(imgIds)
   i=1
   if opt.options == 0:

       for im in images:
           print("{} de {} ".format(i,len(images)))
           im['coco_url'] = "http://cocodataset.org/#explore?id="+str(im["id"])
           img_data = requests.get(im['coco_url']).content
           i=i+1
           with open('images/train2014/' + im['file_name'], 'wb') as handler:
               handler.write(img_data)

   elif opt.options == 1:
       ## For the validation set

       for im in images:
           if i >= 5000:
               break
           else:
               print("{} de {} ".format(i,len(images)))
               im['coco_url'] =
"http://cocodataset.org/#explore?id="+str(im["id"])
               img_data = requests.get(im['coco_url']).content
               i=i+1
               with open('images/val2014/' + im['file_name'], 'wb') as handler:
                   handler.write(img_data)

   else:
       print("Error, that option does not exist")

   print("Download Finished")
   print("Generating the text files")

   d_train =
"/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/dat
a/coco/images/train2014"
   d_eval =
"/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/dat
a/coco/images/val2014"

   for path in os.listdir(d_train):
       path = path + "\n"
       path_train = os.path.join(d_train, path)
       with open("train_1c.txt", "a") as file_object:
           file_object.write(path_train)

   for path in os.listdir(d_eval):
       path = path + "\n"
       path_eval = os.path.join(d_eval, path)
       with open("eval_1c.txt", "a") as file_object:
           file_object.write(path_eval)

   print("Text files generated: train.txt, eval.txt")
```

COPY_IMAGES.PY

Since the API downloaded the images truncated, I had to copy, from the complete datasets, only the images that their id was the same as the ones downloaded by the API, since they were the ones with the people annotations.

```
import shutil,os

#path_train = "images/train2014/"
#path_val = "images/val2014/"

# Using readlines()
file1 = open('train_1c.txt', 'r')
Lines_train = file1.readlines()

file2 = open('eval_1c.txt', 'r')
Lines_val = file2.readlines()

count = 0
# Strips the newline character
for line in Lines_train:
   line_split = line.split("train2014_")[1]
   #print(line_split)
   path_train = "images/train2014/COCO_train2014_{}".format(line_split)
   dest_train = "images_persons/train2014/COCO_train2014_{}".format(line_split)
   print(path_train)
   if os.path.exists(path_train.strip()):
       #print("SI")
       shutil.copy(path_train.strip(), dest_train.strip())
   else:
       print("No existe")
 for line in Lines_val:
   line_split = line.split("val2014_")[1]
   #print(line_split)
   path_val = "images/val2014/COCO_val2014_{}".format(line_split)
   dest_val = "images_persons/val2014/COCO_val2014_{}".format(line_split)
   print(path_val)
   if os.path.exists(path_val.strip()):
       #print("YES")
       shutil.copy(path_val.strip(), dest_val.strip())
   else:
       print("No existe")
```

SNIPPET OF DATASETS.PY

Loads datasets into a PyTorch dataloader to feed iteratively into the model.

```
def create_dataloader(path, imgsz, batch_size, stride, opt, hyp=None,
augment=False, cache=False, pad=0.0, rect=False,
                      local_rank=-1, world_size=1):
    # Make sure only the first process in DDP process the dataset first, and the
following others can use the cache.
    with torch_distributed_zero_first(local_rank):
        dataset = LoadImagesAndLabels(path, imgsz, batch_size,
                                      augment=augment,  # augment images
                                      hyp=hyp,  # augmentation hyperparameters
                                      rect=rect,  # rectangular training
                                      cache_images=cache,
                                      single_cls=opt.single_cls,
                                      stride=int(stride),
                                      pad=pad)

    batch_size = min(batch_size, len(dataset))
    nw = min([os.cpu_count() // world_size, batch_size if batch_size > 1 else 0,
8])  # number of workers
    train_sampler = torch.utils.data.distributed.DistributedSampler(dataset) if
local_rank != -1 else None
    dataloader = torch.utils.data.DataLoader(dataset,
                                             batch_size=batch_size,
                                             num_workers=nw,
                                             sampler=train_sampler,
                                             pin_memory=True,

collate_fn=LoadImagesAndLabels.collate_fn)
    return dataloader, dataset
```

GET_TEXT_FILE.PY

```
import os
d_train =
"/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyT
orch-YOLOv4 (1)/data/coco/images/train2014"
d_eval =
"/Users/rodrigolopezdetoledo/OneDrive/icai/asignaturas/2master/tfm/detectores/PyT
orch-YOLOv4 (1)/data/coco/images/val2014"

for path in os.listdir(d_train):
   path = path + "\n"
   path_train = os.path.join(d_train, path)
   with open("train_1c.txt", "a") as file_object:
       # Append 'hello' at the end of file
       file_object.write(path_train)
```

```
for path in os.listdir(d_eval):
   path = path + "\n"
   path_eval = os.path.join(d_eval, path)
   with open("eval_1c.txt", "a") as file_object:
       # Append 'hello' at the end of file
       file_object.write(path_eval)
reduce_dt.py

from __future__ import print_function

with open("train_1c.txt") as file:
   # loop to read iterate
   # last n lines and print it
   for line in (file.readlines() [0:8000]):
       text_file = open("train_red_1c.txt", "a+")
       print(line, end ="")
       #text_file.write("\n")
       text_file.write(line)
       text_file.close()

with open("eval_1c.txt") as file_val:
   # loop to read iterate
   # last n lines and print it
   for line in (file_val.readlines() [0:1200]):
       text_file_val = open("eval_red_1c.txt", "a+")
       print(line, end ="")
       #text_file_val.write("\n")
       text_file_val.write(line)
       text_file_val.close()
```

GENERATE_ANNOT.PY

Generates the annotations of the Caltech dataset in the correct format for YOLO.

```
import os
import glob
from scipy.io import loadmat

classes = ['person', 'people'] # others are 'person-fa' and 'person?'
squared = True
frame_size = (640, 640) if squared else (640, 480)
number_of_truth_boxes = 0
datasets = {
   'train' : open('train' + ('_squared' if squared else '')  + '.txt', 'w'),
   'test' : open('test' + ('_squared' if squared else '')  + '.txt', 'w')
}
if not os.path.exists('labels'):
   os.makedirs('labels')
```

```
def convertBoxFormat(box):
    (box_x_left, box_y_top, box_w, box_h) = box
    (image_w, image_h) = frame_size
    dw = 1./image_w
    dh = 1./image_h
    x = (box_x_left + box_w / 2.0) * dw
    y = (box_y_top + box_h / 2.0) * dh
    w = box_w * dw
    h = box_h * dh
    return (x, y, w, h)

# traverse sets
for caltech_set in sorted(glob.glob('annotations/set*')):
    set_nr = os.path.basename(caltech_set).replace('set', '')
    set_number = os.path.basename(caltech_set)
    dataset = 'train' if int(set_nr) < 6 else 'test'
    set_id = set_number

    # traverse videos
    for caltech_annotation in sorted(glob.glob(caltech_set + '/*.vbb')):
        vbb = loadmat(caltech_annotation)
        obj_lists = vbb['A'][0][0][1][0]
        obj_lbl = [str(v[0]) for v in vbb['A'][0][0][4][0]]
        video_id = os.path.splitext(os.path.basename(caltech_annotation))[0]

        # traverse frames
        for frame_id, obj in enumerate(obj_lists):
            if len(obj) > 0:

                # traverse labels
                labels = ''
                for pedestrian_id, pedestrian_pos in zip(obj['id'][0],
obj['pos'][0]):
                    pedestrian_id = int(pedestrian_id[0][0]) - 1
                    pedestrian_pos = pedestrian_pos[0].tolist()
                    # class filter and height filter: here example for medium
distance
                    if obj_lbl[pedestrian_id] in classes and pedestrian_pos[3] >
30 and pedestrian_pos[3] <= 80:
                        class_index = classes.index(obj_lbl[pedestrian_id])
                        yolo_box_format = convertBoxFormat(pedestrian_pos)
                        labels += str(class_index) + ' ' + ' '.join([str(n) for n
in yolo_box_format]) + '\n'
                        number_of_truth_boxes += 1

                # if no suitable labels left after filtering, continue
                if not labels:
                    continue

                image_id = set_id + '_' + video_id + '_' + str(frame_id)
                datasets[dataset].write(os.getcwd() + '/images/' + image_id +
('_squared' if squared else '') + '.png\n')
                label_file = open('labels/' + image_id + ('_squared' if squared
else '') + '.txt', 'w')
                label_file.write(labels)
                label_file.close()

                print('finished ' + image_id)

for dataset in datasets.values():
    dataset.close()
```

GENERATE_IMGS.PY

Generates the images of Caltech dataset as PNG, from the original .SEQ format.

```
import os,glob
import cv2 as cv
def save_img(dname, fn, i, frame):
   cv.imwrite('{}/{}_{}_{}.png'.format(
       out_dir, os.path.basename(dname),
       os.path.basename(fn).split('.')[0], i), frame)
out_dir = 'images'
if not os.path.exists(out_dir):
   os.makedirs(out_dir)

def convert(dir):
   for dname in sorted(glob.glob(dir)):
       for fn in sorted(glob.glob('{}/*.seq'.format(dname))):
           cap = cv.VideoCapture(fn)
           i = 0
           while True:
               ret, frame = cap.read()
               if not ret:
                   break
               save_img(dname, fn, i, frame)
               i += 1
           print(fn)
try:
   convert('data/train/set*')
   convert('data/test/set*')
except Exception as e:
   print(e)
```

SQUARIFY.PY

Squares the very rectangular images of Caltech to fit YOLO preferences.

```
import glob
from PIL import Image
frame_size = (640, 640)
directory = 'images'
for frame in sorted(glob.glob(directory + '/*.png')):
   # enlarge 640 x 480 frames up to frame_size
   # by filling the bottom area with whitespace
   new_frame = Image.new('RGB', frame_size, 'white')
   new_frame.paste(Image.open(frame), (0, 0))
   new_frame_path = frame.replace('.png', '_squared.png')
   new_frame.save(new_frame_path, 'png')
   print('saved ' + new_frame_path)
```

TRAIN_ANNOT.PY [56]

Converts CrowdHuman .ODGT format to the preferred format of .TXT files. It is the same for the validation dataset but changing the file names.

```
import numpy
import cv2
from multiprocessing import Process, Manager
import yaml
from tqdm import tqdm
from PIL import Image
intPERSON = 0
intHEAD   = 1
manager = Manager()
return_dict = manager.dict()

def generate_annotations(ii, return_dict):

   line = return_dict[ii]
   del return_dict[ii]
   dictLine = yaml.load(line)
   strID = dictLine['ID']
   image = 'images/train/{}.jpg'.format(strID)

   img = Image.open(image)
   imgWidth, imgHeight = img.size
   # Create .txt label file
   with open('labels/train/{}.txt'.format(strID), 'w+') as txtf:
       for label in dictLine['gtboxes']:
```

56 (https://github.com/alaksana96/darknet-crowdhuman/blob/master/crowdhuman_train_anno.py)

```
     if 'extra' in label and 'ignore' in label['extra'] and
label['extra']['ignore'] == 1:
               continue
     if 'extra' in label and 'unsure' in label['extra'] and
label['extra']['unsure'] == 1:
               continue
           # Person BB
           px = float(label['fbox'][0])
           py = float(label['fbox'][1])
           pw = float(label['fbox'][2])
           ph = float(label['fbox'][3])

           # Head BB
           hx = float(label['hbox'][0])
           hy = float(label['hbox'][1])
           hw = float(label['hbox'][2])
           hh = float(label['hbox'][3])


           # Absolute person BB
           cpx = px + pw/2
           cpy = py + ph/2

           abspx = cpx / imgWidth
           abspy = cpy / imgHeight
           abspw = pw / imgWidth
           absph = ph / imgHeight

           abspx = 1 if abspx > 1 else abspx
           abspy = 1 if abspy > 1 else abspy
           abspw = 1 if abspw > 1 else abspw
           absph = 1 if absph > 1 else absph
           abspx = 0.000001 if abspx < 0 else abspx
           abspy = 0.000001 if abspy < 0 else abspy
           abspw = 0.000001 if abspw < 0 else abspw
           absph = 0.000001 if absph < 0 else absph

           # Absolute head BB
           chx = hx + hw/2
           chy = hy + hh/2

           abshx = chx / imgWidth
           abshy = chy / imgHeight
           abshw = hw / imgWidth
           abshh = hh / imgHeight

           abshx = 1 if abshx > 1 else abshx
           abshy = 1 if abshy > 1 else abshy
           abshw = 1 if abshw > 1 else abshw
           abshh = 1 if abshh > 1 else abshh
           abshx = 0.000001 if abshx < 0 else abshx
           abshy = 0.000001 if abshy < 0 else abshy
           abshw = 0.000001 if abshw < 0 else abshw
           abshh = 0.000001 if abshh < 0 else abshh

           # Write to file
           txtf.write('{} {:.4f} {:.4f} {:.4f} {:.4f}\n'.format(intPERSON,
                                                                abspx,
                                                                abspy,
                                                                abspw,
                                                                absph))

           txtf.write('{} {:.4f} {:.4f} {:.4f} {:.4f}\n'.format(intHEAD,
                                                                abshx,
                                                                abshy,
                                                                abshw,
                                                                abshh))

with open('annotation_train.odgt') as f:
   processes = []
   max_iter = 500
   for ii, line in tqdm(enumerate(f)):
       return_dict[ii] = line
       pcs = Process(target = generate_annotations(ii,return_dict), args = (ii,
return_dict))
       processes.append(pcs)
       pcs.start()
       if ii % max_iter == 0:
           for jj in range(len(processes)):
               processes[jj].join()
           processes = []
   for jj in range(len(processes)):
       processes[jj].join()
   processes = []
```

ANNEX II: INFERENCE RESULTS

In all these folders, there is included detection results in images, in videos and tracking results in those same videos.

YOLOv3:

- COCO: https://drive.google.com/drive/folders/1-vmaHIkTemW_YBu8fBqo9ArkoV9rRvDq?usp=sharing
- KITTI: https://drive.google.com/drive/folders/1G43nnx31stGjMRqUJOQme_5eWI_xIu58?usp=sharing
- CrowdHuman: https://drive.google.com/drive/folders/1IafOwxzG3W9IrrgPE4lM-LgHR3G90CbQ?usp=sharing

YOLOv4:

- COCO: https://drive.google.com/drive/folders/1-95dZhKn-IpUJ6Hr0nShh6eImcCXHpqd?usp=sharing
- KITTI: https://drive.google.com/drive/folders/1bPJP1IITO66mQCJpwMzFVX5ejrCK25bK?usp=sharing
- CrowdHuman: https://drive.google.com/drive/folders/1N9WNfj4TWv7hWX0513vHVLrF5H_fzebx?usp=sharing

YOLOv5:

- COCO: https://drive.google.com/drive/folders/1mg8qWUBgSExFIzX6n52dgMSg5gQMgXzW?usp=sharing

- CrowdHuman: https://drive.google.com/drive/folders/1YrgTPqOGPioVGQxZSZ8Lv4nwc9TcffvT?usp=sharing

ANNEX III: DARKNET VS. ULTRALYTICS

APPROACH

DARKNET APPROACH:

(This is all taken from the authors of YOLOv3 repo & web: https://pjreddie.com/darknet/yolo/)

As seen in the following snippet, this is basically a parser that takes the .CFG file showed in the document and, based on the lines that have indicate which layer it is [yolo], builds the model. It is very impractical and increases the processing time considerably.

```
def create_modules(module_defs, img_size, cfg):
    # Constructs module list of layer blocks from module configuration in
module_defs

    img_size = [img_size] * 2 if isinstance(img_size, int) else img_size  #
expand if necessary
    _ = module_defs.pop(0)  # cfg training hyperparams (unused)
    output_filters = [3]  # input channels
    module_list = nn.ModuleList()
    routs = []  # list of layers which rout to deeper layers
    yolo_index = -1

    for i, mdef in enumerate(module_defs):
        modules = nn.Sequential()

        if mdef['type'] == 'convolutional':
            bn = mdef['batch_normalize']
            filters = mdef['filters']
            k = mdef['size']  # kernel size
            stride = mdef['stride'] if 'stride' in mdef else (mdef['stride_y'],
mdef['stride_x'])
            if isinstance(k, int):  # single-size conv
                modules.add_module('Conv2d',
nn.Conv2d(in_channels=output_filters[-1],
                                                       out_channels=filters,
                                                       kernel_size=k,
                                                       stride=stride,
                                                       padding=k // 2 if
mdef['pad'] else 0,
                                                       groups=mdef['groups'] if
'groups' in mdef else 1,
                                                       bias=not bn))
            else:  # multiple-size conv
                modules.add_module('MixConv2d', MixConv2d(in_ch=output_filters[-
1],
                                                          out_ch=filters,
                                                          k=k,
                                                          stride=stride,
                                                          bias=not bn))

            if bn:
                modules.add_module('BatchNorm2d', nn.BatchNorm2d(filters,
momentum=0.03, eps=1E-4))
            else:
                routs.append(i)  # detection output (goes into yolo layer)

            if mdef['activation'] == 'leaky':  # activation study
https://github.com/ultralytics/yolov3/issues/441
                modules.add_module('activation', nn.LeakyReLU(0.1, inplace=True))
            elif mdef['activation'] == 'swish':
                modules.add_module('activation', Swish())
            elif mdef['activation'] == 'mish':
                modules.add_module('activation', Mish())

        elif mdef['type'] == 'deformableconvolutional':
            bn = mdef['batch_normalize']
            filters = mdef['filters']
            k = mdef['size']  # kernel size
            stride = mdef['stride'] if 'stride' in mdef else (mdef['stride_y'],
mdef['stride_x'])
            if isinstance(k, int):  # single-size conv
                modules.add_module('DeformConv2d', DeformConv2d(output_filters[-
1],
                                                       filters,
                                                       kernel_size=k,
                                                       padding=k // 2 if
mdef['pad'] else 0,
                                                       stride=stride,
                                                       bias=not bn,
                                                       modulation=True))
            else:  # multiple-size conv
                modules.add_module('MixConv2d', MixConv2d(in_ch=output_filters[-
1],
                                                          out_ch=filters,
                                                          k=k,
                                                          stride=stride,
                                                          bias=not bn))

            if bn:
                modules.add_module('BatchNorm2d', nn.BatchNorm2d(filters,
momentum=0.03, eps=1E-4))
            else:
                routs.append(i)  # detection output (goes into yolo layer)
            if mdef['activation'] == 'leaky':  # activation study
https://github.com/ultralytics/yolov3/issues/441
                modules.add_module('activation', nn.LeakyReLU(0.1, inplace=True))
            elif mdef['activation'] == 'swish':
                modules.add_module('activation', Swish())
            elif mdef['activation'] == 'mish':
                modules.add_module('activation', Mish())

        elif mdef['type'] == 'BatchNorm2d':
            filters = output_filters[-1]
            modules = nn.BatchNorm2d(filters, momentum=0.03, eps=1E-4)
            if i == 0 and filters == 3:  # normalize RGB image
                # imagenet mean and var
https://pytorch.org/docs/stable/torchvision/models.html#classification
                modules.running_mean = torch.tensor([0.485, 0.456, 0.406])
                modules.running_var = torch.tensor([0.0524, 0.0502, 0.0506])

        elif mdef['type'] == 'maxpool':
            k = mdef['size']  # kernel size
            stride = mdef['stride']
            maxpool = nn.MaxPool2d(kernel_size=k, stride=stride, padding=(k - 1)
// 2)
            if k == 2 and stride == 1:  # yolov3-tiny
                modules.add_module('ZeroPad2d', nn.ZeroPad2d((0, 1, 0, 1)))
                modules.add_module('MaxPool2d', maxpool)
            else:
                modules = maxpool

        elif mdef['type'] == 'upsample':
            if ONNX_EXPORT:  # explicitly state size, avoid scale_factor
                g = (yolo_index + 1) * 2 / 32  # gain
                modules = nn.Upsample(size=tuple(int(x * g) for x in img_size))
# img_size = (320, 192)
            else:
                modules = nn.Upsample(scale_factor=mdef['stride'])

        elif mdef['type'] == 'route':  # nn.Sequential() placeholder for 'route'
layer
            layers = mdef['layers']
            filters = sum([output_filters[l + 1 if l > 0 else l] for l in
layers])
            routs.extend([i + l if l < 0 else l for l in layers])
            modules = FeatureConcat(layers=layers)

        elif mdef['type'] == 'route2':  # nn.Sequential() placeholder for 'route'
layer
            layers = mdef['layers']
            filters = sum([output_filters[l + 1 if l > 0 else l] for l in
layers])
            routs.extend([i + l if l < 0 else l for l in layers])
            modules = FeatureConcat2(layers=layers)
        elif mdef['type'] == 'route3':  # nn.Sequential() placeholder for 'route'
layer
            layers = mdef['layers']
            filters = sum([output_filters[l + 1 if l > 0 else l] for l in
layers])
            routs.extend([i + l if l < 0 else l for l in layers])
            modules = FeatureConcat3(layers=layers)

        elif mdef['type'] == 'route_lhalf':  # nn.Sequential() placeholder for
'route' layer
            layers = mdef['layers']
            filters = sum([output_filters[l + 1 if l > 0 else l] for l in
layers])//2
            routs.extend([i + l if l < 0 else l for l in layers])
            modules = FeatureConcat_l(layers=layers)

        elif mdef['type'] == 'shortcut':  # nn.Sequential() placeholder for
'shortcut' layer
            layers = mdef['from']
            filters = output_filters[-1]
            routs.extend([i + l if l < 0 else l for l in layers])
            modules = WeightedFeatureFusion(layers=layers, weight='weights_type'
in mdef)

        elif mdef['type'] == 'reorg3d':  # yolov3-spp-pan-scale
            pass

        elif mdef['type'] == 'yolo':
            yolo_index += 1
            stride = [8, 16, 32, 64, 128]  # P3, P4, P5, P6, P7 strides
            if any(x in cfg for x in ['yolov4-tiny', 'fpn', 'yolov3']):  # P5,
P4, P3 strides
                stride = [32, 16, 8]
            layers = mdef['from'] if 'from' in mdef else []
            modules = YOLOLayer(anchors=mdef['anchors'][mdef['mask']],  # anchor
list
                                nc=mdef['classes'],  # number of classes
                                img_size=img_size,  # (416, 416)
                                yolo_index=yolo_index,  # 0, 1, 2...
                                layers=layers,  # output layers
                                stride=stride[yolo_index])

            # Initialize preceding Conv2d() bias
(https://arxiv.org/pdf/1708.02002.pdf section 3.3)
            try:
                j = layers[yolo_index] if 'from' in mdef else -1
                bias_ = module_list[j][0].bias  # shape(255,)
                bias = bias_[:modules.no * modules.na].view(modules.na, -1)  #
shape(3,85)
                #bias[:, 4] += -4.5  # obj
                bias[:, 4] += math.log(8 / (640 / stride[yolo_index]) ** 2)  #
obj (8 objects per 640 image)
```

```
                bias[:, 5:] += math.log(0.6 / (modules.nc - 0.99))  # cls
(sigmoid(p) = 1/nc)
                module_list[j][0].bias = torch.nn.Parameter(bias_,
requires_grad=bias_.requires_grad)
            except:
                print('WARNING: smart bias initialization failure.')

        else:
            print('Warning: Unrecognized Layer Type: ' + mdef['type'])

        # Register module list and number of output filters
        module_list.append(modules)
        output_filters.append(filters)

    routs_binary = [False] * (i + 1)
    for i in routs:
        routs_binary[i] = True
    return module_list, routs_binary
```

ULTRALYTICS APPROACH:

(The script is taken from the creators of YOLOv5 repo: https://github.com/ultralytics/yolov5)

As seen in the following snippet, implementing directly all the layers in PyTorch, there is no need for parsing a 2 thousand-line .CFG file, the only thin needed is a 100 line .YAML defining the architecture and that´s it.

```
class Model(nn.Module):
    def __init__(self, cfg='yolov5s.yaml', ch=3, nc=None, anchors=None):  #
model, input channels, number of classes
        super(Model, self).__init__()
        if isinstance(cfg, dict):
            self.yaml = cfg  # model dict
        else:  # is *.yaml
            import yaml  # for torch hub
            self.yaml_file = Path(cfg).name
            with open(cfg) as f:
                self.yaml = yaml.load(f, Loader=yaml.SafeLoader)  # model dict

        # Define model
        ch = self.yaml['ch'] = self.yaml.get('ch', ch)  # input channels
        if nc and nc != self.yaml['nc']:
            logger.info(f"Overriding model.yaml nc={self.yaml['nc']} with
nc={nc}")
            self.yaml['nc'] = nc  # override yaml value
        if anchors:
            logger.info(f'Overriding model.yaml anchors with anchors={anchors}')
            self.yaml['anchors'] = round(anchors)  # override yaml value
        self.model, self.save = parse_model(deepcopy(self.yaml), ch=[ch])  # 
model, savelist
        self.names = [str(i) for i in range(self.yaml['nc'])]  # default names
        # print([x.shape for x in self.forward(torch.zeros(1, ch, 64, 64))])

        # Build strides, anchors
        m = self.model[-1]  # Detect()
        if isinstance(m, Detect):
            s = 256  # 2x min stride
            m.stride = torch.tensor([s / x.shape[-2] for x in 
self.forward(torch.zeros(1, ch, s, s))])  # forward
            m.anchors /= m.stride.view(-1, 1, 1)
            check_anchor_order(m)
            self.stride = m.stride
            self._initialize_biases()  # only run once
            # print('Strides: %s' % m.stride.tolist())

        # Init weights, biases
        initialize_weights(self)
        self.info()
        logger.info('')

    def forward(self, x, augment=False, profile=False):
        if augment:
            img_size = x.shape[-2:]  # height, width
            s = [1, 0.83, 0.67]  # scales
            f = [None, 3, None]  # flips (2-ud, 3-lr)
            y = []  # outputs
            for si, fi in zip(s, f):
                xi = scale_img(x.flip(fi) if fi else x, si, 
gs=int(self.stride.max()))
                yi = self.forward_once(xi)[0]  # forward
                # cv2.imwrite(f'img_{si}.jpg', 255 * 
xi[0].cpu().numpy().transpose((1, 2, 0))[:, :, ::-1])  # save
                yi[..., :4] /= si  # de-scale
                if fi == 2:
                    yi[..., 1] = img_size[0] - yi[..., 1]  # de-flip ud
                elif fi == 3:
                    yi[..., 0] = img_size[1] - yi[..., 0]  # de-flip lr
                y.append(yi)
            return torch.cat(y, 1), None  # augmented inference, train
        else:
            return self.forward_once(x, profile)  # single-scale inference, train

    def forward_once(self, x, profile=False):
        y, dt = [], []  # outputs
        for m in self.model:
            if m.f != -1:  # if not from previous layer
                x = y[m.f] if isinstance(m.f, int) else [x if j == -1 else y[j] 
for j in m.f]  # from earlier layers

            if profile:
                o = thop.profile(m, inputs=(x,), verbose=False)[0] / 1E9 * 2 if
thop else 0  # FLOPS
                t = time_synchronized()
                for _ in range(10):
                    _ = m(x)
                dt.append((time_synchronized() - t) * 100)
                print('%10.1f%10.0f%10.1fms %-40s' % (o, m.np, dt[-1], m.type))

            x = m(x)  # run
            y.append(x if m.i in self.save else None)  # save output

        if profile:
            print('%.1fms total' % sum(dt))
        return x

    def _initialize_biases(self, cf=None):  # initialize biases into Detect(), cf
is class frequency
        # https://arxiv.org/abs/1708.02002 section 3.3
        # cf = torch.bincount(torch.tensor(np.concatenate(dataset.labels, 0)[:,
0]).long(), minlength=nc) + 1.
        m = self.model[-1]  # Detect() module
        for mi, s in zip(m.m, m.stride):  # from
            b = mi.bias.view(m.na, -1)  # conv.bias(255) to (3,85)
            b.data[:, 4] += math.log(8 / (640 / s) ** 2)  # obj (8 objects per
640 image)
            b.data[:, 5:] += math.log(0.6 / (m.nc - 0.99)) if cf is None else
torch.log(cf / cf.sum())  # cls
            mi.bias = torch.nn.Parameter(b.view(-1), requires_grad=True)

    def _print_biases(self):
        m = self.model[-1]  # Detect() module
        for mi in m.m:  # from
            b = mi.bias.detach().view(m.na, -1).T  # conv.bias(255) to (3,85)
            print(('%6g Conv2d.bias:' + '%10.3g' * 6) % (mi.weight.shape[1],
*b[:5].mean(1).tolist(), b[5:].mean()))

    # def _print_weights(self):
    #     for m in self.model.modules():
    #         if type(m) is Bottleneck:
    #             print('%10.3g' % (m.w.detach().sigmoid() * 2))  # shortcut
weights

    def fuse(self):  # fuse model Conv2d() + BatchNorm2d() layers
        print('Fusing layers... ')
        for m in self.model.modules():
            if type(m) is Conv and hasattr(m, 'bn'):
                m.conv = fuse_conv_and_bn(m.conv, m.bn)  # update conv
                delattr(m, 'bn')  # remove batchnorm
                m.forward = m.fuseforward  # update forward
        self.info()
        return self

    def nms(self, mode=True):  # add or remove NMS module
```

```
        present = type(self.model[-1]) is NMS  # last layer is NMS
        if mode and not present:
            print('Adding NMS... ')
            m = NMS()  # module
            m.f = -1  # from
            m.i = self.model[-1].i + 1  # index
            self.model.add_module(name='%s' % m.i, module=m)  # add
            self.eval()
        elif not mode and present:
            print('Removing NMS... ')
            self.model = self.model[:-1]  # remove
        return self

    def autoshape(self):  # add autoShape module
        print('Adding autoShape... ')
        m = autoShape(self)  # wrap model
        copy_attr(m, self, include=('yaml', 'nc', 'hyp', 'names', 'stride'),
exclude=())  # copy attributes
        return m

    def info(self, verbose=False, img_size=640):  # print model information
        model_info(self, verbose, img_size)
```

It is very obvious why YOLOv5 models are the best performing ones.

ANNEX IV: SUSTAINABLE DEVELOPMENT GOALS

SUSTAINABLE DEVELOPMENT GOALS
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

Figure 128. Sustainable Development Goals

As every project in all day and age, sustainability must be considered for both the outcome and the procedure. Our responsibility does not start and end with the project, its results and its economic benefit, but also with how it impacts society, the planet and those who are not as lucky as us.

Tightly connected with this project, there are three ODS that stand out from the rest. Those are number 3, Good Health and Well-Being, number 4, Quality Education and number 9, Industry, Innovation and Infrastructure. This does not mean that this project cannot impact any of the others, but at this stage, this are the ones more connected to this project.

Regarding Good Health and Well-Being, being the objective of the project to provide a tool to prevent and monitor the spread of Covid-19, the implications in the health and well-being of the society are very clear. Implementing this tool would help to prevent more waves of this pandemic and to help citizens to avoid contracting this virus that has changed completely how the world is contemplated.

Quality education is impacted because all the state-of-the-art methods and technologies that this tool uses. Not only because of that but being part of the solution against this pandemic and useful for so many applications regarding crowd control afterwards, the objective is also to impact the youth and make them more interested in technologies that are already among us but probably not as visible to be noticeable for the younger generation. Hopefully, with more and more people starting to be interested in these technologies, education on technology and innovation become more accessible and more and more people pursue a future in the field.

Regarding Industry, Innovation and Infrastructure, this project hopes to be part of the growing wave of applications in the computer vision field within the data science industry. Using the latest advances in the field to provide a solution not heavily sought and with not that much research behind it, innovation is present through all the project and its application.