

Report ML Visual Object Tracking

The purpose of this report is to outline the improvements made to my tracking system by comparing the improvements with the baseline from the practical session. The report will first describe the baseline, then discuss the enhancements made:

1. **Adding a New Detector (YoloV8 nano):** This section will talk about why and how we added YoloV8 nano, a new and better detector, to the system. It will explain how this change has made object detection more accurate and efficient.
2. **Managing Tracks with an Age-Based Approach:** Here, the report will describe a new way of managing tracks by looking at their 'age' or how long they have been in the system. This method helps in keeping track of objects more consistently and accurately over time.
3. **Making Search Strategies Better by Filtering Bounding Boxes:** This part will discuss how we improved the system by making the search for objects smarter. It will explain how filtering out less important bounding boxes (the boxes that show where objects are on the screen) makes the tracking more accurate and reduces mistakes.

Additionally, the report will mention the use of Trackeval, a tool that gives us numbers to measure how much the system has improved.

Baseline presentation

The baseline established during the practical sessions serves as the default comparison point for evaluating other models. This baseline comprises the following key features:

- **Multi-Object Tracker Using IOU:** The system utilizes Intersection Over Union (IOU) with bounding boxes to track multiple pedestrians. This method involves comparing the overlap between predicted bounding boxes and ground truth boxes to track the movement of each pedestrian across frames.
- **Pre-generated Detection Data:** The tracker operates on detection data read from a CSV file. This data, pre-generated by an external detector, outlines the location and dimensions of bounding boxes for each pedestrian in each frame.
- **Hungarian Algorithm for Assignment:** The system employs the Hungarian Algorithm for assigning detections to existing tracks. This optimization algorithm ensures that each detection is matched with the correct track, minimizing overall costs associated with incorrect matches.
- **Kalman Filter for Prediction and Correction:** The Kalman Filter is utilized for predicting the future state of each pedestrian and then correcting these

predictions with new measurements. This addition significantly enhanced the system's ability to deal with the uncertainties in pedestrian movements, notably reducing the number of identity switches (IDs) from approximately 200 to a substantially lower figure.

- **Visualization of Tracking Results:** The system provides visual results by overlaying the bounding boxes of the tracks on the video frames. This visual representation not only makes it easier to understand the tracking performance but also helps in qualitatively assessing the accuracy of the tracker.

Adding visual features with a CNN

I wanted to find out how much better it is to use Intersection Over Union (IoU) together with features from a Convolutional Neural Network (CNN), even though we had to use them in our class project. So, I chose to compare this method with other changes to see if it really works better.

Tracker architecture

The pictures below show the new things we added to the diagram from our class project. We put these new parts in color so you can see them easily. The usual parts like the Hungarian Algorithm and the Kalman filter are still there and haven't changed. The main setup looks a lot like before, but we put in some new blocks at different steps in the process.

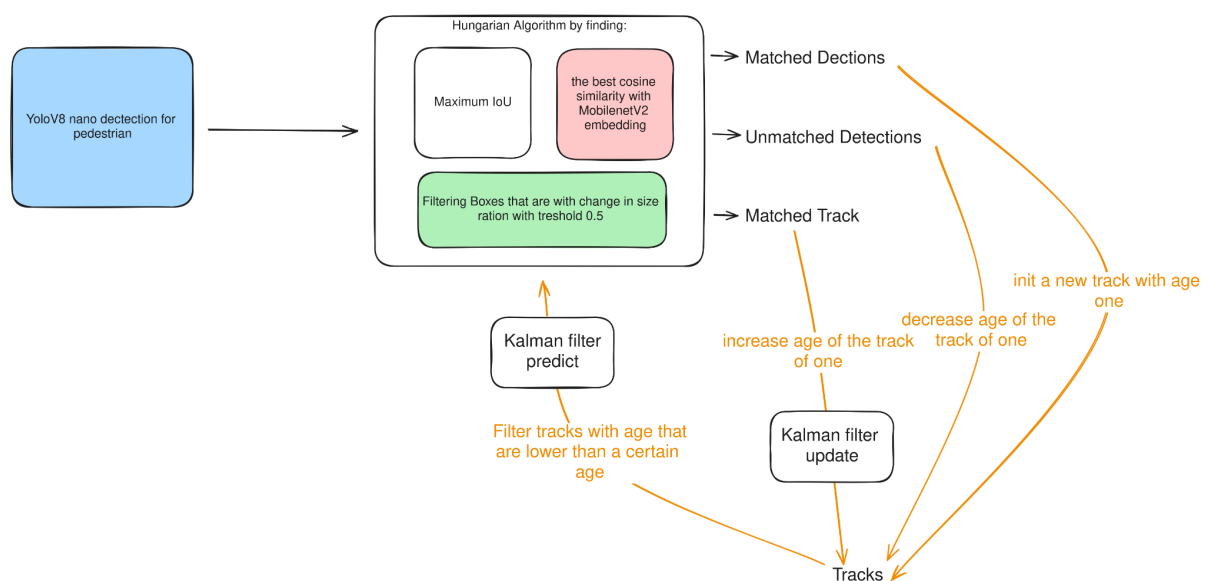






Figure 1: Architecture schema of the tracking system

Description of the schema

-  Adding a New Detector (YoloV8 nano)
-  Embedding similarity with MobilenetV2
-  Track Management system with age
-  Efficient Search Strategies with boxes filtering

Improvements

Next, we'll talk about the new things we added to the basic version and explain how we put them in, what good things they were supposed to bring to the tracker, and what challenges they might have. We'll also talk about the hard parts we faced when we were adding these features.

Feature embedding with a CNN model

The baseline relies completely on Intersection over Union, this means that it relies completely on geometric information without taking into account the visual similarity between the two bounding boxes.

$$similarity = \alpha * IoU$$

Initially, I used a method to calculate the similarity between a detection and a track. There are several ways to measure visual similarity between models, but in this instance, a CNN model was chosen. I am most comfortable with PyTorch, which offers a straightforward method for preprocessing images and extracting the embedding layer by removing the final layer, instead of the classification layer. To utilize a CNN model for computing visual similarity, I followed these steps:

1. **Preprocessing the Image:** This step involves making the image compatible with the CNN, which may involve padding or cropping.
2. **Computing Embeddings:** The CNN computes the embeddings for each bounding box.
3. **Calculating Cosine Similarity:** Next, the cosine similarity between the embeddings of two bounding boxes is calculated.

The choice of the right model was crucial. Initially, I considered using Resnet50, but since I wanted to run the entire algorithm on my laptop without a GPU, I found the inference with Resnet50 to be slow. Therefore, I opted for MobileNetV2, a smaller model with 3.5 million parameters, compared to Resnet50's 23 million. However, it

was still slow. Upon further investigation, I realized that I hadn't batched the different bounding boxes together at each frame. Neural networks are optimized to work with batches, and making this change significantly improved the computation time of my model.

Despite these optimizations, this enhancement remained the most time-consuming aspect of my algorithm. YoloV8 nano, used with the ultralytics library, offers fast inference, aligning with the company's goal of minimizing architecture size and maximizing inference speed. However, MobileNetV2 still required more computing power, despite having a similar model size. A potential improvement could involve using a smaller model or exploring ways to enhance the inference speed of a CNN model.

It's now possible to consider visual similarity between a detection and a track. For combining these similarities, I chose alpha equals 0.5 and beta equals 0.5, which provided balanced and satisfactory results. A further enhancement could involve performing a hyperparameter search across multiple videos and using quantitative metrics to find the optimal balance between different approaches.

$$similarity = \alpha * IoU + \beta * cosine_similarity$$

The code was organized in tracking/features.py for modularity and ease of replacement. It's encapsulated in a class called DeepFeatures, which I utilize in the tracking process during the computation of similarity.

Adding a new detector (YoloV8)

I chose this improvement because I was very interested in seeing how a new detector would affect the overall tracking system. As we learned in our lectures, the detector is a key part of the tracking system. The previous tracks were flickering and not always accurate.

The original detector used pre-detected bounding boxes with another tracker. During the lectures, we learned to read a CSV file and manage detections frame by frame. I wanted to create an algorithm that would handle content in real time. So, in my work with Yolo, I decided to process the detection at each frame. This had its pros and cons.

Pros:

- I could use any video with pedestrians, and my algorithm would work on it. This gave me the chance to test my implementation on various scenarios.

Cons:

- Processing each frame can be slow because it takes a lot of computing power. But I found a solution for this. I tried different YoloV8 models with various numbers of parameters. In the end, I chose YoloV8 nano because it runs really fast on my laptop and still gives very good results. The trade-off between the model size and the reduced speed was really small, probably because the task at hand, which was the detection of pedestrians, is a fundamental part of Yolo model's training.

The main advantage as we look at the visual result is that the detection seems really accurate and that the blinking effect with no detection between some frames for the object seems to disappear. In the code I decided to change some hyperparameters such as keeping some confidence threshold and I had the ability to use a much higher threshold because the detection was more accurate. Another interesting observation is that the movements of different people were tracked very well. For example, if a pedestrian's bounding box became bigger due to the movement of their arm, the tracking remained highly accurate. Overall, the number of total identifiers was drastically reduced by about a factor of 10, which had a significant impact.

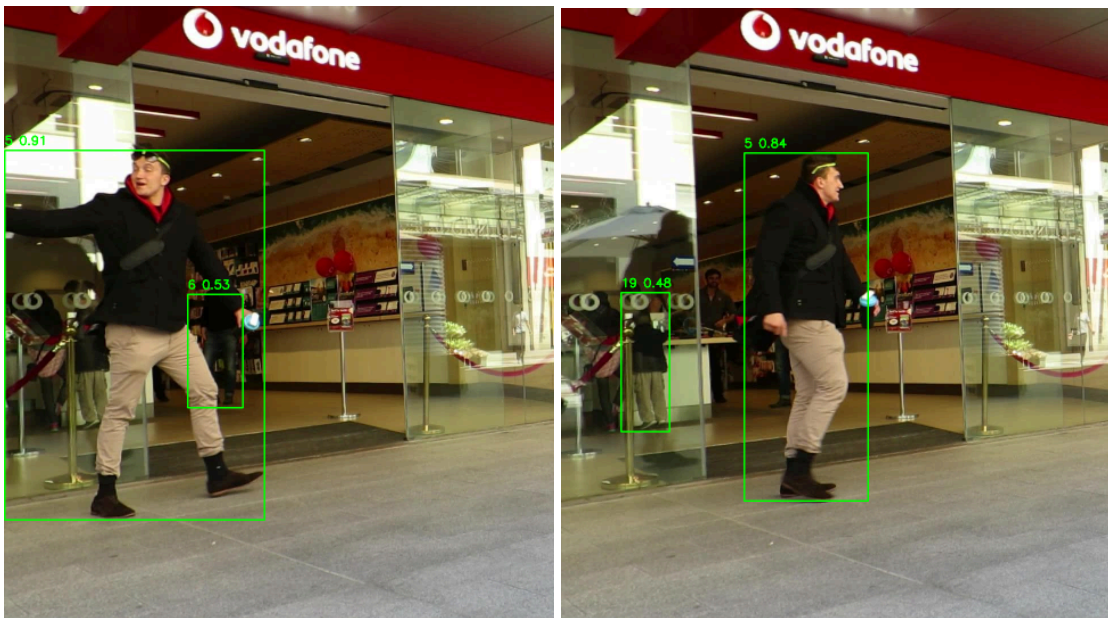


Figure 2: example of bounding box deformation with YoloV8

I created a YoloDetector class in tracking/detector.py. This class has several methods to initialize and detect pedestrians on the street. I opted to use the supervision library to parse the different detections, as it makes the code highly

reusable and modular. Integrating the detector posed a challenge, as it was quite different from the previous one. A significant portion of the components were tightly integrated, so replacing the detector required some effort to separate and reorganize the detection components within the tracking file. This approach of modularization and code reusability was also applied to the other improvements.

Efficient Search Strategy: Filtering Based on Bounding Box Size

The similarity of different bounding boxes is now calculated using both geometric and visual approaches. These similarities are then matched using the Hungarian method, optimizing the overall process. However, we can still improve how we search for different bounding boxes, as not all boxes need to be matched. A simple method to consider is examining the size of the bounding boxes.

To compare the sizes of two bounding boxes, we use a size ratio. If this ratio exceeds a certain threshold, we choose not to match that detection in the subsequent steps. This approach not only yields better results but also helps prevent children from switching IDs.

Integrating this was not overly complicated. It was directly incorporated into the tracker. Thanks to the previous step requiring the code to be divided, the Tracker class in `tracking/tracker.py` became very user-friendly and easily adaptable for adding new features. While this improvement might not have provided the most significant value compared to others, it did offer a way to avoid some incorrect matches and made the search for similarities between bounding boxes more efficient.

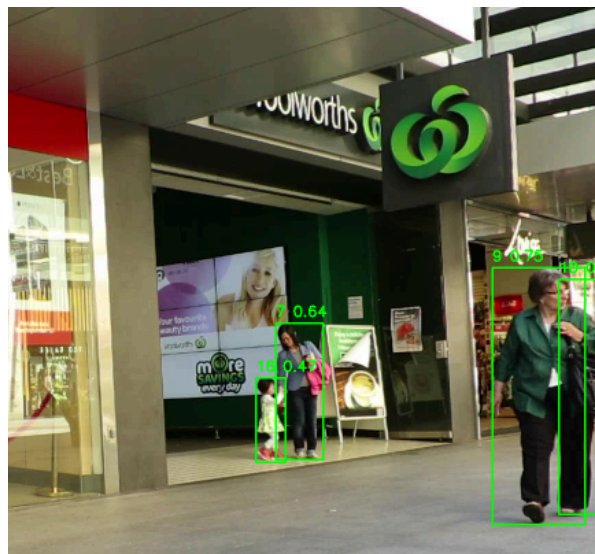


Figure 3: Example of Improved Tracking with a Bounding Box That Previously Alternated Between the Mother and Child

Age-Based Approach for Track longevity

The goal of this method was to make track management better by not getting rid of a track too quickly. We gave each track an 'age' to help with this. Here's what we wanted to do:

1. **Keep Tracks Longer:** If a track has been around for a while in the video, we try to keep it. This helps avoid the problem of tracks that pop up just for a moment and then disappear.
2. **Deal with Objects Blocking Each Other:** Sometimes, one object might block another in the video. With this method, if a track has been around for a while (it has a high age), we keep it even if we can't see it for a bit.

Here's how the system works:

- **Starting a New Track:** We begin a new track with an age of 1.
- **Updating a Track:** Every time we see the same track again, we add 1 to its age.
- **Removing a Track:** If we miss a track, we reduce its age.

This improvement was not too hard to put in because our Tracker class is set up in a way that's easy to change and add things. It made things a bit better, especially in the video ADL-Rundle-6, where it seemed like the IDs stayed the same for people longer. We can check this with numbers and data.

The hard part was picking the right numbers for the system, like how much age a track should lose if we don't see it. For the ADL-Rundle-6 video, I chose -3, and it seemed to work well. But maybe we can find even better numbers by trying different ones on more videos. That could be something to make it better in the future.

Benchmarking

The goal of this part is to benchmark the different improvements made to the algorithm in terms of different metrics using the TrackEval benchmark on the MOT15 challenge. The goal of this section is to have different not only visual metrics that are subjective but quantitative metrics that are objective.

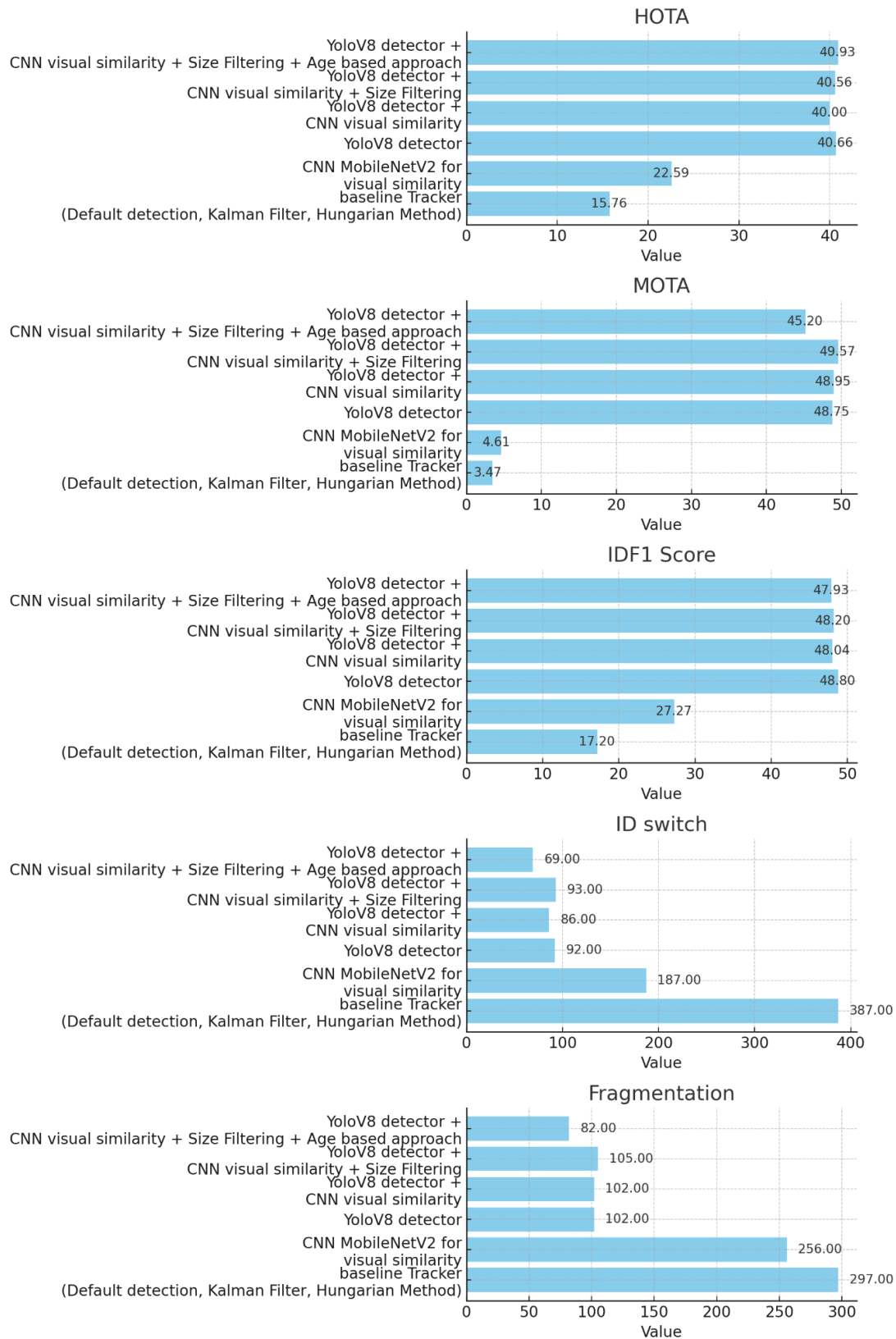
Benchmarking quantitative result

The metrics that will be used in this benchmark are the following ones on the ADL-Rundle-6 sequence for the detection of pedestrian:

1. **HOTA (Higher Order Tracking Accuracy):**
 - HOTA checks how well the tracking system finds objects and keeps track of them correctly. and It makes sure the system is good at both finding objects (detection) and knowing which object is which over time (association).
2. **MOTA (Multiple Object Tracking Accuracy):**
 - MOTA looks at the whole picture of tracking by considering three things: when the system misses objects, when it thinks there are objects that aren't there (false positives), and when it mixes up the IDs of the objects. MOTA is a metric that measures the mistake that the model made.
3. **IDF1 Score:**
 - IDF1 measures how good the system is at giving the same ID to the same object across different frames in the video.
4. **ID_Switch (Identity Switch):**
 - ID_Switch counts how often the system gets confused and changes the ID of an object from one frame to the next. Fewer ID_switch mean that the system has less confusion between objects.
5. **Fragmentation:**
 - Fragmentation counts how many times a track stops and then starts again. When the model loses track of an object and then finds it again and treats it like a new object.

Model Description	HOTA	MOTA	IDF1 Score	ID switch	Fragmentation
baseline Tracker (Default detection, Kalman Filter, Hungarian Method)	15.758	3.4737	17.203	387	297
CNN MobileNetV2 for visual similarity	22.593	4.6117	27.265	187	256
YoloV8 detector	40.655	48.752	<u>48.795</u>	92	102
YoloV8 detector + CNN visual similarity	40.004	48.952	48.038	86	102
YoloV8 detector + CNN visual similarity + Size Filtering	40.56	<u>49.571</u>	48.204	93	105
YoloV8 detector + CNN visual similarity + Size Filtering + Age based approach	<u>40.93</u>	45.199	47.928	<u>69</u>	<u>82</u>

Benchmarking metrics interpretation



HOTA metric

For HOTA, putting in the CNN to see how similar things look did make things better. But what really made a big difference was using the YOLO detector. Even just using YOLOv8 by itself made things a lot better. You could see this when you looked at the video too. This means that these models are really good at keeping track of the same objects and spotting them right. Other changes didn't do much for this metric. It looks like the first way we were detecting objects wasn't as good as what YOLO does. Also, we used the small YOLO model, called nano.

MOTA metric

For the Multi-Object Tracking Accuracy, or MOTA, we look at three kinds of mistakes: when the system thinks there's something that isn't there (false positives), when it misses something it should have seen (missed targets), and when it gets mixed up and changes the IDs of the objects (identity switches). This metric kind of tells the same story as the one before. The YOLOv8 model is way better than the other models. But this time, making the system look at the size of things (size filtering) did help a bit. It made the MOTA score go up by almost 2 points. This is probably because it stopped some wrong detections from matching with tracks.

IDF1 Score

The IDF1 score checks if the system can give the same ID to the same thing in different parts of the video. Adding the CNN and looking at what things look like helped the score go up by 10 points. The YOLOv8 model also made the IDF1 score much better when it was put in. But it's weird, mixing YOLOv8 with CNN didn't make things as good as just using YOLOv8 alone. Since the CNN takes up a lot of computer power, maybe it's better to just use YOLOv8. The other changes didn't really do much once the detector was switched.

ID Switch

ID switches are bad – we want as few as possible. The fewer ID switches there are, the better the system is working. In my project, all the improvements I made really helped cut down on ID switches. Sure, we might have lost a little in other areas, but objects got tracked for longer without swapping IDs so much. It's sometimes hard to get the balance right between the different scores. It all depends on what you're trying to do. In my case, using the CNN to check how things look really helped a lot. It brought the ID switches down from around 200 to just 69 when I combined it with the other changes I made.

Fragmentation

Fragmentation counts how many times a track starts and stops – it's linked to ID switches because a new ID means a new track. But for my project, YOLOv8 had a bigger effect on reducing fragmentation, which is something we want to keep low. Also, when I added the age-based approach, the results got even better. The CNN made a difference too, but it wasn't as big of a deal for fragmentation as it was for reducing ID switches.

Conclusion

The various enhancements demonstrated significant improvements compared to the baseline, particularly the change of the detector, which had a profound impact on both subjective and objective metrics. Trackeval provided a quantitative means to compare this model. The CNN showed improvements on its own, but these were less pronounced when combined with YOLOv8. Other enhancements, such as the refined track matching algorithm and the size-based efficient filtering method, offered modest improvements for certain metrics. According to the MOT-Challenge, the results achieved by my model using trackeval seem accurate but are specific to a particular video.

In conclusion, observing the tracker's evolution with each improvement made this project a learning experience. It was insightful to witness the gradual development and the various stages of a multi-object tracker. The project necessitated multiple code rewrites to ensure a modular design, facilitating the easy addition of new components. There is still potential for further enhancements, particularly in better handling occlusions and predicting pedestrian directions using techniques like LSTM.

