Robust object tracking for inventory monitoring systems

TUM Data Innovation Lab (TUM-DI-LAB)

Munich Data Science Institute (MDSI)

Technical University of Munich

&

PreciBake GmbH

Authors	Ying Gu, Jingjie Guo, Sarah El Beji, Benjamin Bangert, Alper Kagan Kayali, Aras Bayrakceken
Mentor(s)	M.Sc. Mathias Sundholm, M.Sc. Maximilian Schreil, M.Sc. Alexander Dolokov (PreciBake GmbH)
Project lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (Board of Directors of MDSI)

Feb 2022

Abstract

Object tracking is a technology used in different sectors, such as the food industry. While object tracking is beneficial in many cases, such as inventory monitoring, it also comes with several problems; one of which is containing many failure cases. These cases include occlusions, frame drops, or re-identification of the objects. Furthermore, food items might leave and reenter the scene, which needs a re-identification on that item. While the problems are easy to understand, solutions to these problems are not trivial. In fact, Simple Online and Realtime Tracking algorithm (SORT) preserves and computes the tracking information of the objects in the scene; however, they do not compute the appearance information of the inventory. Since failure cases such as occlusion and frame drops are related to the appearances of the objects, this information should be processed to overcome these cases. We propose that if we can include appearance information in our algorithm, we can develop a robust one to solve many of these cases. Towards this end, we investigate different modes of failure of the object tracking technology. We simulate occlusion, frame drop, and re-identification cases using a 3D Computer Graphics software called Blender. Finally, we offer three different algorithmic solutions to these issues, which are PixProSORT, PixProSORT with cached feature vectors, and DeepSORT + PixPro. We also evaluate these solutions with HOTA and its sub metrics and assess the results based on this evaluation. We compare these solutions with SORT. We conclude that although each proposed solution is better at certain use cases, PixProSORT with cached feature vectors performs better than others for the re-identification case. Finally, we provide future work for the proposed algorithms, such as improving the simulation dataset on modes of failure, enabling them to work in a real-life inventory monitoring scenario.

Contents

1 Introduction

Object tracking is one of the research areas of Machine Learning (ML) where the algorithm tries to estimate the trajectory of an object as it moves around a scene [**1**]. It is a challenging problem, due to several reasons such as altering visuals of the objects in the scene or instantaneous movements that might be caused by the camera [**2**]. Although there are several challenges, it is a research field that has been utilized in many different industries such as motion-based recognition or surveillance. One of these research areas is inventory monitoring, which is a technology to be used to track the flow of goods and the assets of the facility [**3**]. Inventory monitoring can utilize different aspects of the assets in the scene, such as the size of the inventory, or the age. In addition, it can also track the number of goods in the inventory via object tracking. While the object tracking technology is helpful, there are many issues that might come with it. One of these issues is the noise, which corrupts the data, that is created when the data is being saved. Another issue might be the illumination due to a bright light source. Other problems can be seen in Figure 1. While the effects of some of these issues started to disappear thanks to better camera technology, some of them remain to be an issue. One example to these problems that remains is object occlusion; because no matter how the camera technology is improved, it is a challenge for the algorithms to capture an object that is occluded. Furthermore, several datasets are designed, such as PETS [**4**] or KITTI [**5**], to assist ML researchers to discover new algorithms to solve the remaining issues. Although these datasets are helpful, they are not adequate if one wants to investigate a specific problem; since these issues occur differently in every tracking scenario.

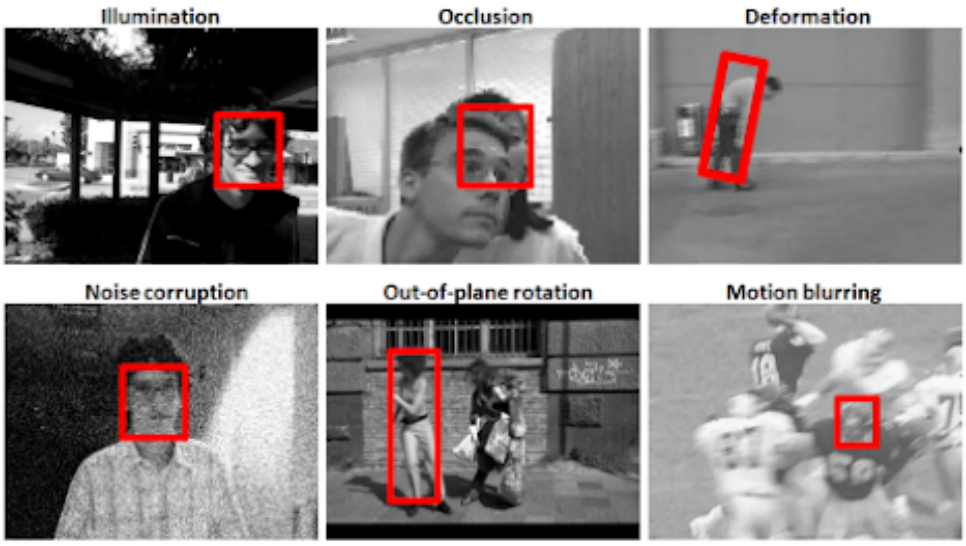

Figure 1: Different Object Tracking Issues [**6**].

One of the fields of inventory monitoring is food tracking, which includes its own set of issues alongside the problems that inventory monitoring has. Food tracking is crucial to a facility in the food industry, since using object detection is not feasible, or affordable for them since their inventory changes quickly. It is difficult for a food tracking algorithm to track objects with similar appearance information since the algorithm tends to confuse two of those objects as given in Figure 2; which is one of the problems that food tracking contains. It is also cumbersome for the algorithm to track the age of the inventory, which is a crucial factor in food tracking. Finally, it also lacks a specific food tracking dataset that would be helpful to solve these issues.

We propose three different algorithms to tackle general object tracking problems that exist in the food industry. These algorithms are PixProSORT, PixProSORT with cached feature vectors, and DeepSORT + PixPro, and as the names of these algorithms suggest, these algorithms are modifications of the baseline. To overcome the problems special to food tracking, we propose a newly simulated dataset that covers a variety of food objects. This dataset enables us to evaluate our proposed solutions with respect to the issues related to food tracking.

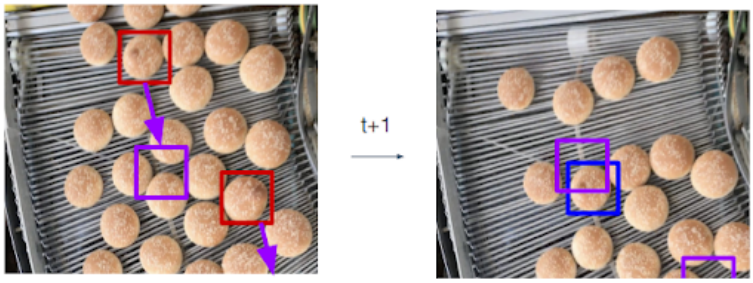

Figure 2: A Food Tracking Example.

2 Background

2.1 SORT

To enable online inventory monitoring an algorithm that is very lightweight and robust is needed. SORT developed by Bewley et al [**7**] fulfills a lot of the criteria needed for a suitable tracking algorithm for inventory monitoring. The algorithm is able to work with an update rate of 260Hz while maintaining comparable accuracy to state-of-the-art online trackers.

The SORT algorithm is lightweight because the authors reduced the amount of information used to track objects. The object tracking is done by modeling a linear velocity model which only takes the bounding box of a detection as an input. SORT is a tracking-by-detection framework that solves the problem of multiple object tracking [**7**] and is divided into different parts: object detection, velocity estimation model, data association and tracking.

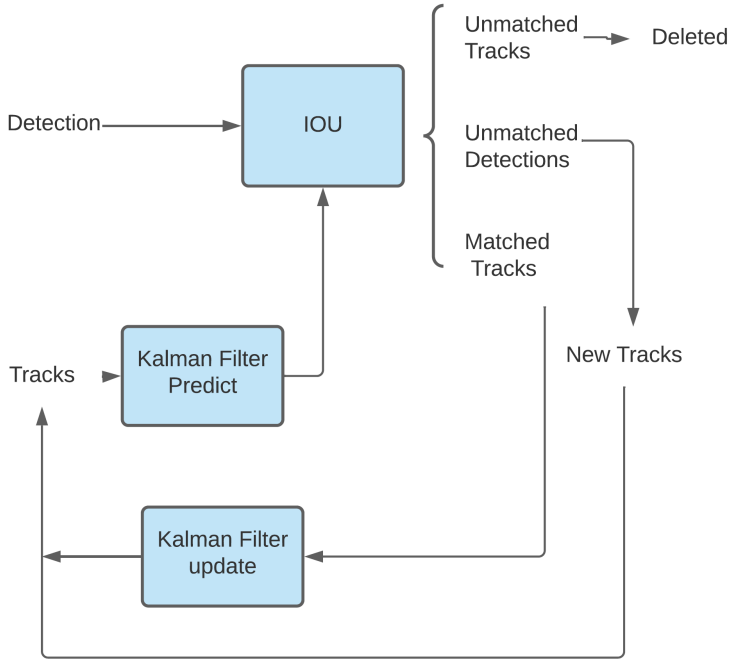

Figure 3: Flowchart of the SORT algorithm.

Detection The authors of SORT evaluated the model performance for different detectors and decided to use the Faster Region CNN (FrRCNN) detection framework [**8**]. The FrRCNN has two stages where in the first stage features are extracted and regions of interest (ROI) are proposed and in the second stage the object in the ROI is classified.

Velocity Estimation Model The SORT algorithm uses a linear constant velocity model which approximates the inter-frame displacement of each object. The bounding boxes of detected objects $o_{i,t}$ are uniquely described by the horizontal and vertical center coordinates $u_{i,t}, v_{i,t} \in \mathbb{N}$, the aspect ratio r_i which is constant over time and the area $s_{i,t}$, here i denotes detection index at time t for $i, t \in \mathbb{N}$. A combination of i and t uniquely determines a detected object and it's position. The state of a track $T_{k \in \mathbb{N}} \in \mathbb{T}_t$, where $\mathbb{T}_t$ is the set of all tracks at time $t \in \mathbb{N}$, is modelled as:

$$x_{k,t} = [u_{i,t}, v_{i,t}, s_{i,t}, r_i, u^{'}_{i,t+1}, v^{'}_{i,t+1}, s^{'}_{i,t+1}]^T,$$

where it is assumed that $o_{i,t}$ is the detected object last assigned to T_k The velocity components of the model are then solved optimally with a Kalman filter framework. The predicted bounding box at time $t+1$ of the object i at time t is then defined by $u^{'}_{i,t+1}, v^{'}_{i,t+1}$ and $s^{'}_{i,t+1}$.

Data Association and Tracking To correct these predicted bounding boxes the correct bounding box has to be associated with the track. By iterating from time point t to $t+1$ the associated bounding boxes are found by applying the Hungarian Algorithm to solve the bipartite matching problem for $\mathbb{T}_t$ and $\mathbb{D}_{t+1}$, the set of detected objects at time

$t+1$. The cost matrix is given by

$$cost_{n,m} = IOU(x'_{n,t+1}, o_{m,t+1}),$$

here $x'_{n,t+1}$ be the predicted bounding box of T_n at time t, $o_{m,t+1}$ be a detected object at time $t+1$ and IOU determines the intersection over union. A minimum IOU is introduced to reject assignments where the overlap is less than $IOU_{min} = 0.3$.

A new Track T_k at time $t+1$ is created when finding a detected object $o_{m,t+1}$ which satisfies $cost_{n,m} < IOU_{min}$ for all n. T_k is then initialised with the objects bounding box and the velocity is set to zero with large covariance values to reflect its uncertainty.

A Track will be terminated if there happened no data association for t_{LOST} frames. The authors of SORT chose $t_{LOST} = 1$.

Modes of Failure The low amount of information needed to perform the tracking task makes the algorithm lightweight but it also leads to major drawbacks when applying it for inventory monitoring.

The out of the box SORT algorithm is not able to re-identify objects, which can lead to problems in inventory monitoring where objects will leave and reenter the camera view. Also while workers maintain the inventory there is no way to prevent partial and full object occlusion.

While also being lightweight the linear velocity estimation model is not able to grasp, with respect to frame rate, complex or fast motion of objects. When objects in the inventory are moved by workers or machines the velocity model might not be able to model the motion performed by workers which can by default be very complex and unpredictable for such a simple approach.

2.2 Datasets

Since this project aims to improve existing methodologies and compare novelty ideas to established methods, a standardized dataset is needed to test and evaluate our models. We have looked into several datasets, which provide multiple sequences of ground truth data with bounding boxes for multiple object tracking: PETS, KITTI, Detrac, MOTChallenge [**6**].

We have decided to use mainly the MOTChallenge dataset since it is well standardized and provides a large collection of sequences. Another important factor was that it provides detections for all the sequences, which can be used in our methods whenever needed.

The authors also provide evaluation tools that can be used to compare novelty methods to the existing ones while also providing sub-metrics such as recall and precision. A downside was that it mainly focuses on pedestrian tracking while our project is focused on inventory tracking. But we found this to be the case for most of the existing methods, and evaluation on pedestrian data is necessary for comparison to established object tracking methods.

2.3 Simulation

As mentioned in chapter 2.1 the limitations of SORT and evaluating their extent are crucial when applying it for inventory monitoring. Since the potential datasets from the previous chapter may cover object occlusion, objects leaving and reentering the view or having fast-moving objects, with respect to the frame rate, it still is not possible to focus solely on the evaluation of the extent of SORT's limitations.

To circumvent the problem of finding suitable test data we use Blender which is a 3D modeling and rendering software, we make use of it as a data simulation platform. This has the advantage that we can simulate data such that it triggers the modes of failures of SORT in a controlled environment. Using this approach makes detailed studying of the algorithm possible and shines a light on the strengths and weaknesses when performing inventory monitoring with SORT for certain scenarios.

Using such a data simulation platform not only allows flexibility in the amount and quality of data but has also a controlled environment that can be quickly changed and reused for future experiments without the need for human relabeling.

2.4 Evaluation

Since an important part of the project is to compare novelty methods to existing ones a standardized method for multi-object tracking evaluation was necessary. The problems with existing evaluation methods include; lack of standard tools, lack of evaluation results of existing object tracking methods, over-emphasis of detection or association. To this end, we have decided to use the HOTA [**10**] metric, based on the fact that it is tailored to equally emphasize detection and association, and aligns with human visual evaluation. Another important factor is that the HOTA metric also provides sub-metrics which allow us to look into the precision and recall, or association and detection of the models separately and identify the problems with ease. The authors also provide standard tools to easily evaluate standard formats of tracking data.

Sub-metric space of HOTA follows the lines of Detection and Association, and Recall and Precision. Whereas the detection metrics are intuitive and are used as they are already established in other works, a high association recall and a high precision mean, in this context, the tendency of the model to break the trajectory of a single object into smaller trajectories of multiple objects, or fusing the trajectories of multiple objects into a single longer trajectory of a single object, respectively. Other than these metrics HOTA also provides a localization error, which shows the spatial similarity of the detections with the ground truth.

3 Simulation

As discussed above, we generate our own scenes using Blender. We choose Blender because it is a free and open-source 3D computer graphics software toolset used for creating animated 3D models. Blender also supports python scripting. The python scripts we

developed are automated such that no blender manipulation is needed to reproduce the rendered scenes and such that re-running the same script gives different scenes.

3.1 Blender

Blender environment A scene in Blender contains different 3D objects that can be animated along a timeline. The mesh objects (unlike camera and light objects) in the scene have different properties: the name in the scene, the mesh (vertices, edges, faces of the object), the world transformation matrix (translation, rotation, scale), to name a few important.

Rendering in Blender A camera is placed such that the camera view is what will be rendered as simulated images. Once a scene is created, the Eevee renderer -a rendering engine included in Blender- renders the frames of each scene with settings of our choosing (image resolution, frame range, image format, etc). The images in our data consist of the rendered frames.

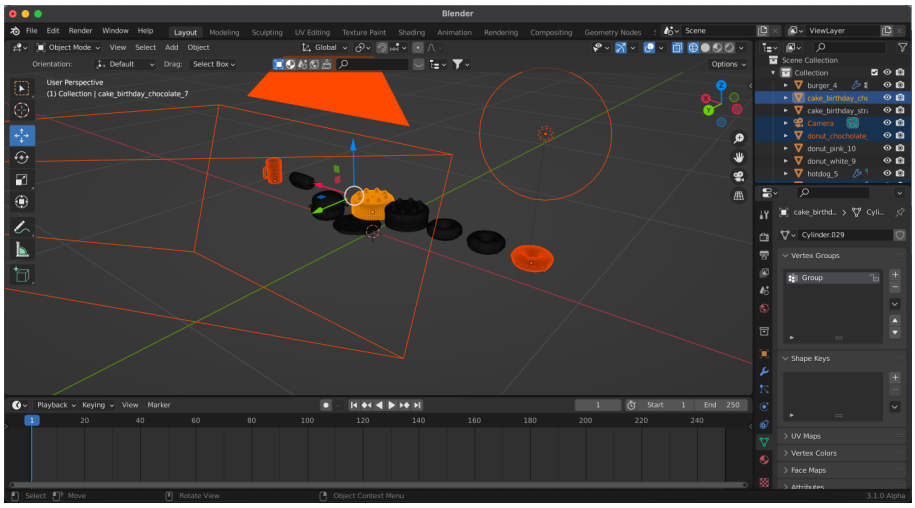

Figure 4: Blender Viewport with selected mesh objects (mug, cake, donut, etc), light and camera. The timeline shows the range frame and potential animations' key-frames.

Figure 5: Rendered image using Eeevee.

Figure 6: Camera view in Blender.

Assets and BlenderKit The scenes we simulate using Blender must have different objects to be tracked since we are interested in multiple object tracking. They also must have different natures, and be varied in size, color and other visual features. In fact, we want to test models that rely on visual features for tracking. For time efficiency and to stay in the scope of our project, we do not impose on ourselves to create the single objects from scratch: we use pre-modeled objects from BlenderKit.

Blenderkit is a partially free shared library with an open community. Creators upload models that can be downloaded for free (58% of the dataset) with no limit on the number of downloads. These models can be added to scenes and contain all data about the used textures, the objects meshes, .. Thus, we use free models in relation to the food industry downloaded from BlenderKit. Based on them, we create a common assets file of 21 objects which are saved as assets objects in a blend file. This file is then used to load the necessary objects for each scripted scene. We use the asset browser to manage object creation.

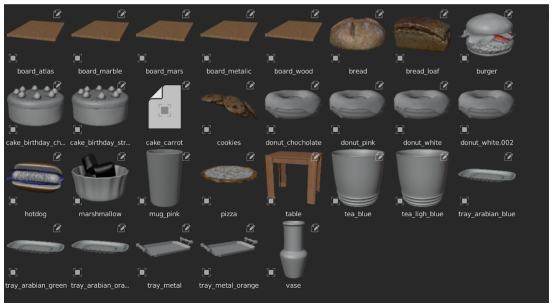

Figure 7: Created asset browser with 21 objects based on projects from BlenderKit. The issue of texture shows here.

Figure 8: Example of a rendered frame using objects from the asset created browser. The texture problem does not show here.

Limitations of BlenderKit BlenderKit, obviously, alleviates our workload and the quality of the free objects is quite high. We could also change the texture and colors of some objects to look visually different and thus enrich more our assets file. However, BlenderKit's use is still limited. Firstly, we still needed to clean the models manually (change gravity centers, delete dependencies in model collections, join objects ..). Secondly, we joined different parts of one model as one object to facilitate our scripting for tracking. However, when joining detailed meshes with less detailed ones, we lose the credibility and realism of the final object. We had to not use models having this problem and thus we had an even smaller number of objects created (since we only worked with free models which was already a small set). Finally, texture data was unstable.

3.2 Dataset

For tracking multiple objects, we need the images where animated multiple objects appear, a ground truth file that gives the tracks of each object visible in each frame and a detection file that contains the tracks of each object visible in each frame with bounding boxes

modified to be close to what a designed detector would give. For ground truth and detection files, We respect the file structure used in the MOT15 [**9**] dataset.

3.2.1 Ground Truth

Extraction of bounding boxes In Blender, 3D objects' vertices are projected on the camera's view space. Thus, for each vertex in a 3D object's mesh, we can access its 2D projection as well as the depth of each vertex to the camera (z-depth). A negative z-depth means the object is behind the camera. Therefore, constructing a bounding box for an object is the same as constructing a bounding box of its projected vertices on the camera screen. Our objects are projected in a realistic manner ("perspective" projection).

z-depth Some literature states that the z-depth does not return depth information of a vertex but the distance to the camera. We do a sanity check to verify if we need to use another information to capture the depth of a vertex [**11**]. For the Eevee renderer, the z-depth is the correct one we need. It is crucial to make such a check because z-depth is needed to identify which of two objects that have intersecting bounding boxes is occluding the other.

Total, partial and no occlusion When an object is only slightly occluded, a detector should still be able to detect it. Therefore, we define three levels of occlusion: total partial and no occlusion based on the percentage of occluded area p_i of an object Obj_i's bounding box. Thresholds imposed on this percentage give the three-level occlusion decision. To calculate the percentage of total occluded area p_i of an object Obj_i, we proceed as follows:

- We identify potential occluding objects Occ_j of Obj_i:
  - The bounding boxes of Occ_j and Obj_i must intersect.
  - Occ_j's z-depth must be lower than Obj_i's (closer to camera).
- When many bounding boxes intersect with Obj_i's bounding box and also intersect between each other, it is necessary to add the intersection areas only once. By intersection, we mean the intersection between bounding boxes and the areas are the areas of bounding boxes.
- We normalize the total occluded area by Obj_i's bounding box's area.

Each track must contain the right bounding box. Thus, when an object is completely occluded, we do not include it in the ground truth as track ($p_i > 0.7$). When the object is not occluded ($p_i < 0.3$), we add it as a track and its bounding box is its complete bounding box. If the bounding box is partially occluded ($p_i \in [0.3, 0.7]$), then we need to choose a new bounding box for only the non-occluded part of the object Obj_i 9. In order to do that :

- We calculate the smallest biggest bounding box containing all the intersections of Obj_i and its occluding objects. This box is contained in the Obj_i's total bounding box.

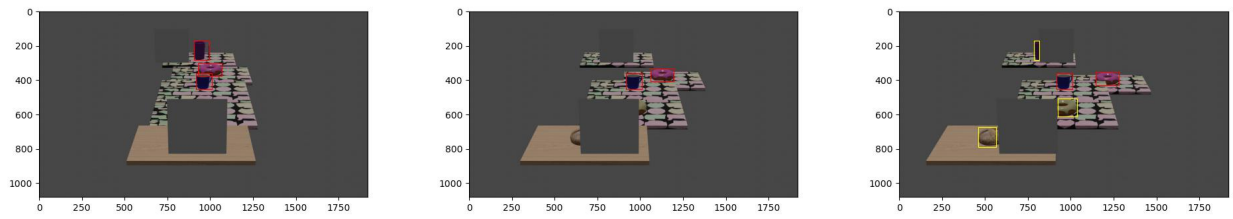

Figure 9: An example of bounding boxes: Red boxes are full bounding boxes. Yellow boxes are partial bounding boxes. The object Obj_i is the bread behind the occluding paper Occ_1. In (a), the bread does not have a bounding box (total occlusion). In (b), although a part of the bread shows, p_i is still too big ($p_i > 0.7$ to consider partial occlusion). In (c), p_i is low and we add a track for the bread with the yellow bounding box.

- The partial bounding box that is affected to the track of Obj_i is the one that gives the highest non-occluded area.

3.2.2 Simulate a Detector

The simulation of data as described above serves as ground truth data. To simulate the output of a representative detector there is still the need to add noise to the simulated data. This is done by removing bounding boxes with a probability 0.05 to simulate a false negative detection. Furthermore, bounding boxes are added at random positions with random size with a probability 0.05 to simulate a false positive detection. Last but not least the shapes and positions of bounding boxes are changed with random noise to assert that for the moving objects the simulated detector has different qualities of detections. The probabilities were chosen to simulate the model performance of FrRCNN [**8**] used by the SORT authors.

3.3 Failure Cases

Different failure scenes have been created using the tools that have been defined previously. All of these scenes had laid on five main foundations:

- All of them were compatible with the ground truth extraction algorithm that generates labels suitable to the proposed algorithms in this report.
- All of them were created using the assets that are defined here.
- All of the assets in the scene followed a certain track that is provided to them.
- All of the scenes included both full and partial occlusion.
- All of these scenes included some sort of randomness in them, meaning that each scene that is created is different from the others; enabling variety in the dataset.

Failure Cases have been divided into two categories: Occlusion With a familiar object and Occlusion with a foreign object.

3.3.1 Occlusion With A Familiar Object

In this failure case, the occluding object is familiar to the Food Tracking field. In other words, the occluding object in this scene is an object that might appear in a real-life scenario such as a donut, bread, etc. While the occluding object is stuck, other objects are mobile. The mobile objects are occluded while moving in the upper right direction. In order to create the scene, 11 different objects are selected randomly from the asset library. 10 of these objects are assigned as mobile while 1 of them is assigned as the occluding one. By the nature of the movement, some of the objects are fully occluded while some of them are partially occluded. This failure case is explored in three different scenarios:

- Occlusion Without Any FPS Change: This scene covers the failure case explained above without any FPS Change in it.
- Occlusion With Frame Drops: This scene covers the same failure case with FPS Changes in it.
- Occlusion With Lag: This scene covers the same failure case with including laggy objects.

(See Appendix A for the scene)

Occlusion Without Any FPS Changes This scene is created by the failure case given above. In order to create this scene, 40 frames are generated. The velocity of the objects does not change and since there are no FPS drops or lags in the scene, steady velocity is also observed by the human eye.

Occlusion With Frame Drops This scene is created by the same failure case, but it contains frame drops. The frame drops are obtained by not recording every second frame for some time. This approach is applied for 10 frames, meaning that only 5 of these frames are recorded. Since 5 frames are lost, 35 frames are generated in this scene. In addition, the velocity of the objects does not change. However, the human eye might perceive them as they are accelerating in the frame drop time frame.

Occlusion With Lag This scene is created by the same failure case, but it contains lag. The lag is obtained by just recording one frame out of 10 frames. Since only 1 frame is recorded out of 10, 31 frames are generated in this scene. The velocity of the objects does not change in this scene as well. The human eye can observe the lag.

3.3.2 Occlusion With Foreign Object

While occlusion with familiar objects is the general case for failure cases, the objects in the scene might also occlude with foreign objects. While the reason for a foreign object's appearance is unclear, it might still appear. In order to investigate this occlusion, two different scenarios are proposed:

- Occlusion with a Palette Object
- Occlusion with a Cube Object

Occlusion with a Palette Object In this scenario, 13 different objects are selected randomly from the asset library. They are placed orderly on top of a dinner table. In addition, a palette object occludes one of these objects. The occlusion is removed gradually, enabling one to investigate full and partial occlusions.
(See Appendix B for the scene)

Occlusion with a Cube Object In this scenario, two objects that are familiar two the real-life scenario are occluded by a cube object. However, this scenario is especially different from the other ones. While the camera is stable in all of the previous failure cases, it is rotating in this one. By the nature of the rotation, full occlusion and partial occlusion can be observed in the scene.
(See Appendix B for the scene)

4 Tracking Algorithms with Appearance Information

SORT makes use of only motion information for association. Using appearance information in the assignment matrix can solve the issues mentioned in Modes of Failure. In this section, a common workflow of a multiple object tracking algorithm using appearance features is introduced. A model named PixPro [**16**] is analyzed if it can be utilized to be the feature extractor in the workflow. Subsequently, three modified methods are presented, in which the pretrained PixPro model is integrated as the extractor. In the first model, the assignment matrix of the SORT model is calculated depending on feature representations of detections instead of the IOU method. In the second method, the feature representations of tracks are cached in memory, so that an object can be identified after occlusion. In the last modified DeepSORT model, pretrain of deep association metrics is omitted. Instead, feature information is extracted and stored for each detection for the Cascade algorithm. For each method, a short discussion of encountered challenges is explained.

A common workflow of a multiple object tracking algorithm using appearance features is shown in Figure 10. The first step is to run a detector to obtain bounding boxes of objects. In the second step, a model extracts visual features of the bounding boxes. Then we can get a matrix by computing the similarity or the distance among those features. This matrix is used as the assignment matrix for the Hungarian algorithm. The bounding boxes in the current frame are matched to the most similar ones in the previous frame. In this way, a track for one object is maintained until it doesn't appear in a certain number of frames.

4.1 PixPro

Unsupervised learning has shown good performances in extracting visual representations using contrastive learning [**12**]. Most of the previous works focused on instance-level pre-

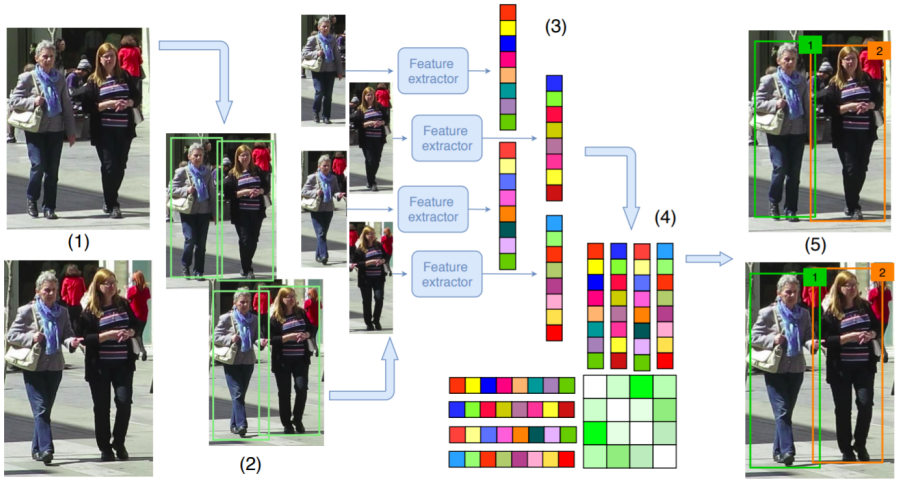

Figure 10: Workflow of a multiple object tracking algorithm using appearance features [**1**].

text tasks, while the PixPro model introduces pixel-level pretext tasks for learning dense visual features [**16**].

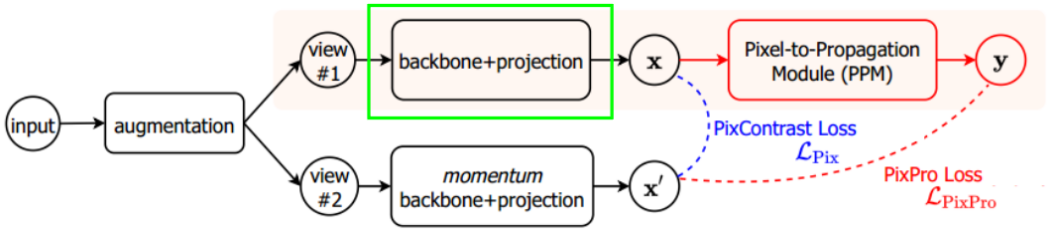

Figure 11: Architecture of the PixPro model [**16**].

The pixel-level contrastive learning means each pixel is treated as a single class and the model is aimed to distinguish different pixels. As shown in Figure 12, two random crops are obtained in an image, containing some intersection part. Those pixels whose distances are within a threshold in two views are treated as positive pairs, while pixels with large distances in two views are treated as negative pairs. We get two feature maps x and x' after those two views are fed into the architecture in Figure 11. A contrastive loss is computed based on the feature vectors of positive and negative pairs, where cosine distance is used to get the similarity between two feature vectors of two pixels.

Furthermore, the PixPro model takes pixel-to-propagation consistency into consideration. The contrastive learning method only encourages spatial sensitivity, which discriminates spatially close pixels into different labels. This helps to make a prediction in boundary areas between different labels accurate. But spatial smoothness is also very important. Spatial smoothness encourages spatially close pixels to be similar since close pixels tend to belong to the same label. In order to propagate the features of surrounding similar pixels to a pixel, a pixel-to-propagation module(PPM) is added into the architecture. The PPM

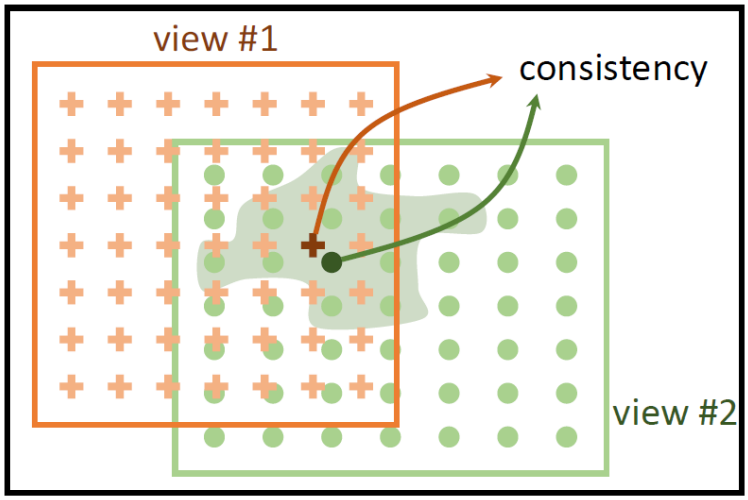

Figure 12: An illustration of the PixPro method, where two views are randomly cropped from an image(outlined in black) [**16**].

computes the similarities between other pixels and the target pixel x and propagates the features of other pixels to this target pixel x using the similarities as weights:

$$y_i = \sum_{j \in \Omega} s(x_i, x_j) \cdot g(x_j) \tag{1}$$

where $j \in \Omega$ means all pixels in the same view as x_i. The similarity function s(. , .) is cosine distance, namely dot product of two feature vectors. The Pixpro loss is computed as:

$$\mathcal{L}_{PixPro} = -cos(y_i, x'_j) - cos(y_j, x'_i) \tag{2}$$

where pixel i and pixel j are a positive pair in the original image. x'_i and y_i are feature vectors of pixel i computed from two different pipelines as shown in Figure 11. x'_j and y_j are obtained for pixel j in the same way as pixel i.

The features of the corresponding pixels in two crops of the same image are encouraged to be consistent. In this manner, we don't have to label anything manually and we can assume the result can generalize to other objects. That's the reason we pick up the PixPro model to extract visual features with no need to train the model to specific objects.

We want to use the PixPro model as the feature extractor in the workflow in Figure 10. To evaluate the PixPro model for extracting visual features, we do some visualization to show the performance of it.

Pixel Level Visualization:

The output feature map x in the upper pipeline in Figure 11 contains 256 channels. The Euclidean norm of the feature vector for each pixel in this feature map is already normalized to 1 in the PixPro model. So we can directly compute the dot product between two

pixels to get the similarity between them.

As shown in Figure 13, we take the center point of the person with a red T-shirt as an example and compute the dot product of this point with all other pixels in the feature map. The result is shown on the right side of Figure 13. The yellow point in the center is the feature vector we take. We can see a rough shape of a person on the left side of the feature map, which corresponds to the person in a black T-shirt in the original image. There is a red wall behind the person in the black T-shirt. But the feature vector of the person with a red T-shirt is more different from the red wall than the person with a black T-shirt. It indicates that the visual features extracted by PixPro are invariant to color. We also notice that the floor in the feature map is well distinguished from people. If a relative bad detector is used, a false bounding box on the floor is very unlikely to be matched to a bounding box of a person, since they are very different in the feature map of PixPro. The result shows that the feature vectors of pretrained PixPro contain semantic information and we can use them to compute similarities among objects.

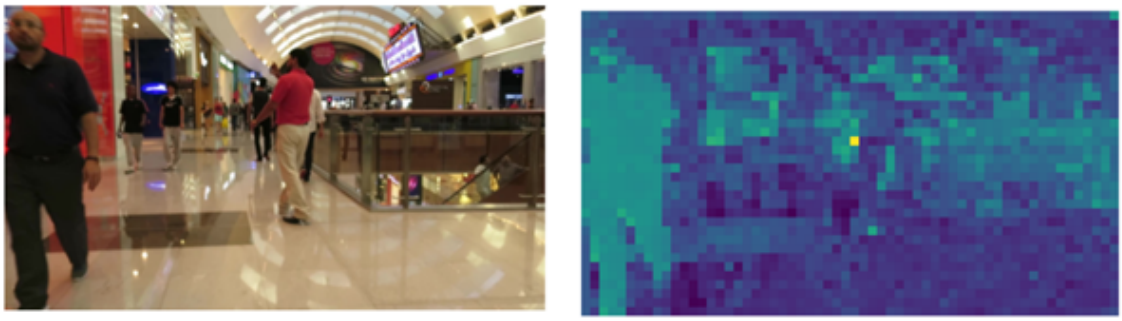

Figure 13: Visualization of PixPro in pixel level.

Bounding Box Level Visualization:

We take two not consecutive frames containing objects with the same ids to do the visualization in bounding box level. The way is very similar to step (4) in Figure 10. The feature maps are acquired by feeding frames into the PixPro model. For each bounding box, the feature vector is the average of the feature vectors in the bounding box and then normalized to a vector with the Euclidean norm of 1. Then an assignment matrix is calculated using the dot product of feature vectors of corresponding bounding boxes. This matrix is visualized in Figure 14. It's clear that objects with id1, id2 and id4 are most similar to themselves in another frame. But the object with id3 is similar to id1 in another frame almost as much as it is to itself. This visualization shows that PixPro is not 100 percent reliable. The Hungarian algorithm can still deal with the case in Figure 14 since it minimizes the overall cost, which is the negative value of this similarity matrix. But PixPro can lead to unreliable results when the scene is more complicated with more objects in it.

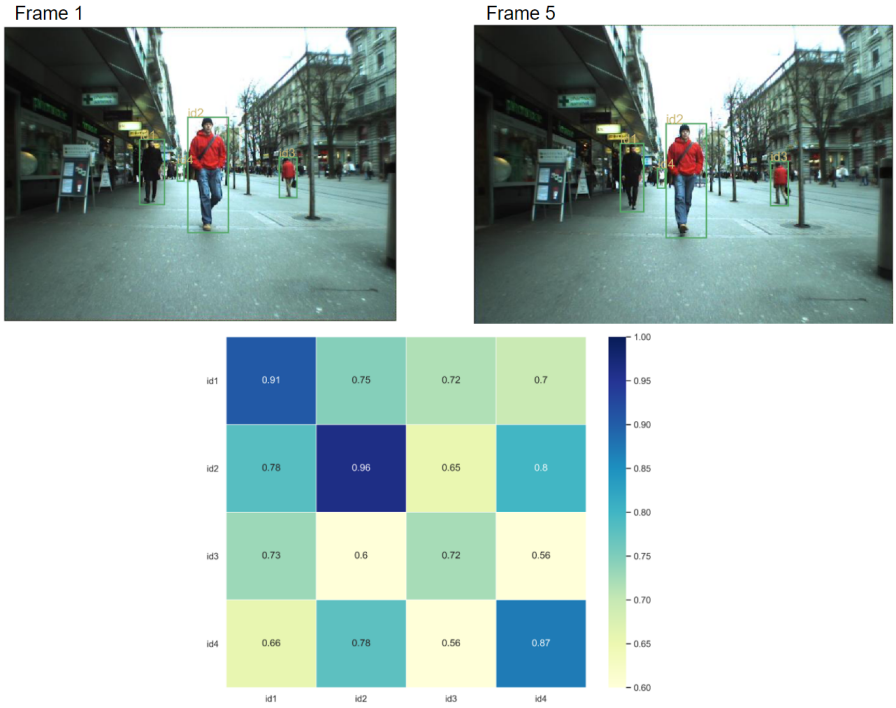

Figure 14: Assignment matrix for bounding boxes in two frames, in which objects with same ids are detected. In the heatmap, the horizontal axis represents frame 1 containing objects from id1 to id4. The vertical axis represents frame 5 containing the objects with the same ids. The bigger the values are, the more similar objects are in terms of visual features.

4.2 Integrating PixPro in SORT model

As aforementioned, the SORT algorithm unitizes correction information from associating objects in two adjacent frames. The Kalman filter then uses object location in frame t+1 as a new measurement for the object in frame t to update the states. In our proposed PixProSORT model PixPro algorithm replaces IOU as the association criterion. The flowchart of PixProSORT is shown in Figure 15.

As mentioned in the PixPro model combines pixel-level pretext tasks for learning dense feature representations, which combines contrastive learning at the pixel level and a pixel-to-propagation consistency task. The pretrained PixPro model can be applied at the pixel level for learning visual representations [**16**]. As a downstream task of PixPro, the pretrained encoder network is utilized to extract image features, which composed of a backbone network adopting ResNet network and projection, which consists of two successive 1x1 convolution layers with a batch normalization layer and a ReLU layer in-between to produce image feature maps of a certain spatial resolution. The part of pretrained encoder is marked in the green box in Figure 11.

During the processing, the pretrained extractor reads each frame and outputs a feature vector with a depth of 256. The spatial resolution varies depending on the resolution of the input frame. For an input frame with the size of 640x480x3, the output shape of the

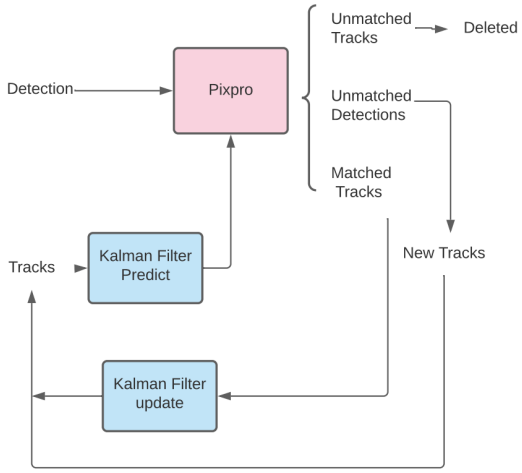

Figure 15: Flowchart of PixProSORT model. The core Algorithm "Intersection of Union" in the SORT algorithm is replaced by PixPro method (box filled with pink color).

feature vector is 1x256x20x15. Given the coordinates of bounding boxes, the weighted mean pixel value of each bbox is computed and stored. This process is illustrated in Figure 16. For instance, in a frame with three detections and four predictions bounding boxes, after the above process, We get two feature vectors of size 3x256 for detections and 4x256 for tracks. The two feature vectors are then passed to calculate using cosine distance, which is a method used by search engines and can be calculated by the inner product of two normalized vectors. In this case, the output has a size of 3x4. The next step is the implementation the Hungarian Algorithm same as in the SORT algorithm.

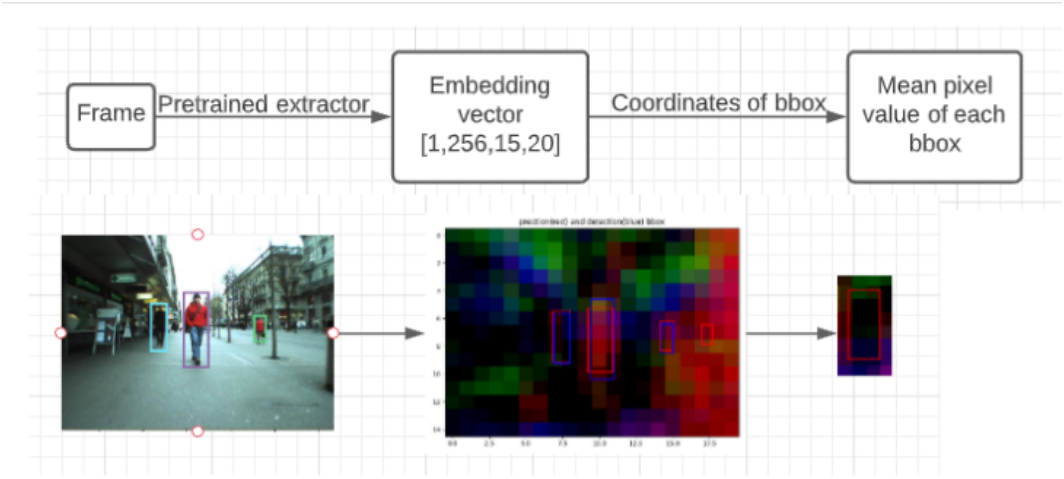

Figure 16: Procedure of computing feature representation of bounding boxes in the modified model.

There are different methods to find the pixel-level representation for a bounding box. For example, we can use the pixel value of the middle point of a bbox. This method is suitable for objects which do not change their form during movement (e.g. tracking cars) or

objects with unique colors. In our demo video, PixProSORT is applied to tracking pedestrians by computing the weighted mean pixel value. Each tile pixel value is weighted by its area, then they are summed up and divided by the whole bbox area (Figure 17). The height and width of target detection influence the choice of compute method. The feature representation of small detections in the PixPro model may be downsampled to even just one pixel. Thus, different factors should be taken into account when choosing the computational methods.

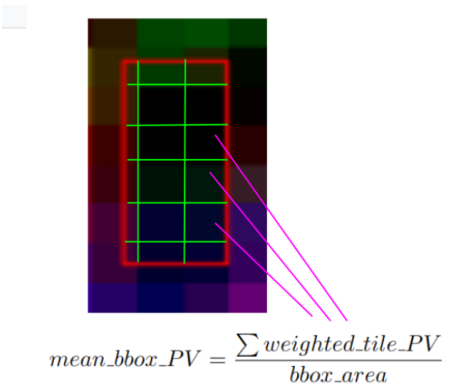

Figure 17: Computing weighted mean pixel value of a bbox.

In our demo video of MOT15 ETH-Bahnhof [**15**] training data, a pedestrian with a red jacket located in the middle of the image can't be tracked starting from frame 13 and his index is switched after several frames (Figure 18). For this test, the method visualized in Figure 17 is utilized. In further trials with the same datasets, we only change the computing method for the bounding box identification. The pedestrian with a red jacket can be tracked without interruption.

Figure 18: Pedestrian with red jacket fails tracking.

The phenomenon of losing tracking in the middle of an image appears also in other datasets. This leads to arise doubt if the parameters of the extractor in the region give a good feature representation. In addition the rightmost region of the image in the middle

of the bottom half of Figure 16 is colored with red. This image visualized the output of the feature vector of the image on the left side. From the real image, we can observe that the rightmost region is not specific but rather similar to the background region.

Another way to improve the performance of this model is to use a different metrics learning method to measure the similarity. If the target object is large, the combination of IOU and PixPro to calculate the cost metrics may also be reasonable. If the object is small, the method is not suitable, since IOU lays a big burden on the initial states of the Kalman filter.

The computing time of the modified model on the test dataset, which has a pixel value of 640x480, is quite acceptable. But for datasets with high resolutions, the computing time extends. The model is then not suitable for real-time tracking.

4.3 PixProSORT with Cached Feature Vectors

Unlike vanilla SORT or vanilla PixProSORT, PixProSORT with cached feature vectors does not delete unmatched tracks immediately. Each track has an attribute storing its age, namely the number of frames since the last time this track is successfully matched to a detection. The age of a track will be reduced to 0 if it's matched to a detection and updated by Kalman Filter. If the age of this track is bigger than the value max_age, this track will be considered to already leave the scene and deleted. If the age of an unmatched track is smaller than max_age, the track can still go through the Kalman Filter prediction process.

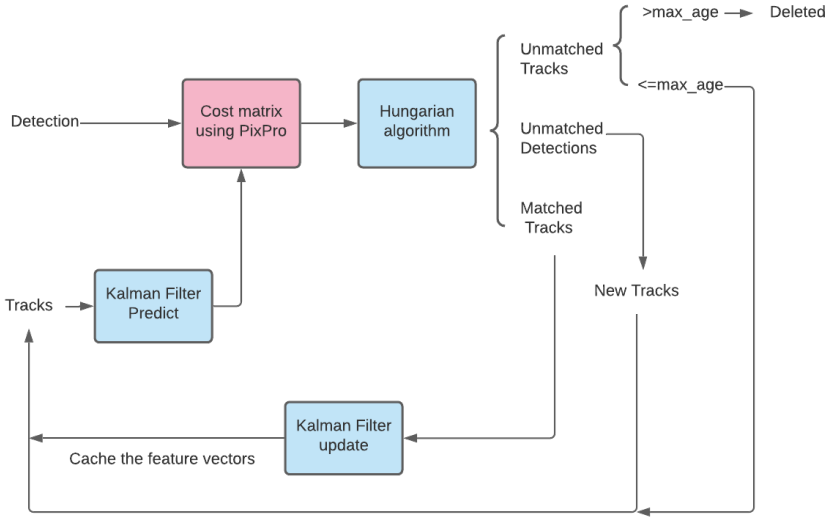

Figure 19: Flowchart of PixProSORT with cached feature vectors. The unmatched tracks are not deleted until they don't appear in the number of maximum age. The feature vectors of matched bounding boxes are cached in memory for re-identification.

Another difference from the vanilla PixProSORT is that this algorithm caches the feature vectors of bounding boxes in the track. After a matched track is updated by Kalman

Filter, the feature vector of the bounding box in the current frame is cached in this track. During the calculation of the assignment matrix, we take the smallest value of the dot product between the feature vector of a detection and cached feature vectors of a track. The smallest value means the most similar appearance of the cached bounding boxes in this track to the current detection. If a track is occluded for a few frames, its cached feature vectors can help it to match to current detection after it appears again in the scene. In this manner, we can reduce the frequency of id switches, which is a big problem for vanilla SORT and vanilla PixProSORT.

4.4 Integrating PixPro in DeepSORT model

The DeepSORT algorithm integrates appearance information to improve the performance. The model can better handle the situation where the target is blocked for a long time, which is solved by core algorithm cascade [**7**]. The appearance feature is extracted by the depth correlation metric pre-trained on the large-scale pedestrian re-cognition dataset, which needs to be learned offline in advance, and its function is to extract distinguishable features. So the association can be made based on feature similarity besides overlap. Metric learning aims to construct an embedding where two extracted features corresponding to the same identity are likely to be closer than features from different identities. In the bottom half Figure 20 flowchart of DeepSORT is visualized. Detailed procedure of combination of the two key algorithms "Matching Cascade" and "IOU" can be checked here [**7**]. The boxes filled with yellow color describe the offline training of a deep association metric feature representation of the original DeepSORT algorithm used a novel cosine metric learning [**14**]. Each object in the datasets for DeepSORT composes two parts in a line. The first part stores the basic information such as the number of frames, index and the location of bounding boxes. As a second part, the feature representation of the object is then followed.

Since the pretrained extractor in PixPro model outputs the dense feature information of an image, we can integrate it into DeepSORT. So that the pretrain of deep association metrics can be omitted. Unlikely the datasets for original DeepSORT model, only the basic information of each object is required. This means datasets of MOT can be directly employed for testing. The PixPro model is integrated into "Matching Cascade" algorithm, which is visualized in Figure 20 with the green filled box. During the process, each frame is passed through PixPro model and outputs a feature vector. Given the coordinates of a bounding box the mean pixel value of the object is then computed, it is subsequently saved as its feature for this object for further processing. The whole modification is altered in the function of "create_detection" in the original DeepSORT model.

The advantage is obvious, the offline pretrain of deep association metrics for different customer data is not the precondition for utilizing the DeepSORT model. Thence the performance of the modified model depends on the PixPro model. As a result, it has the same challenges described in the previous sections.

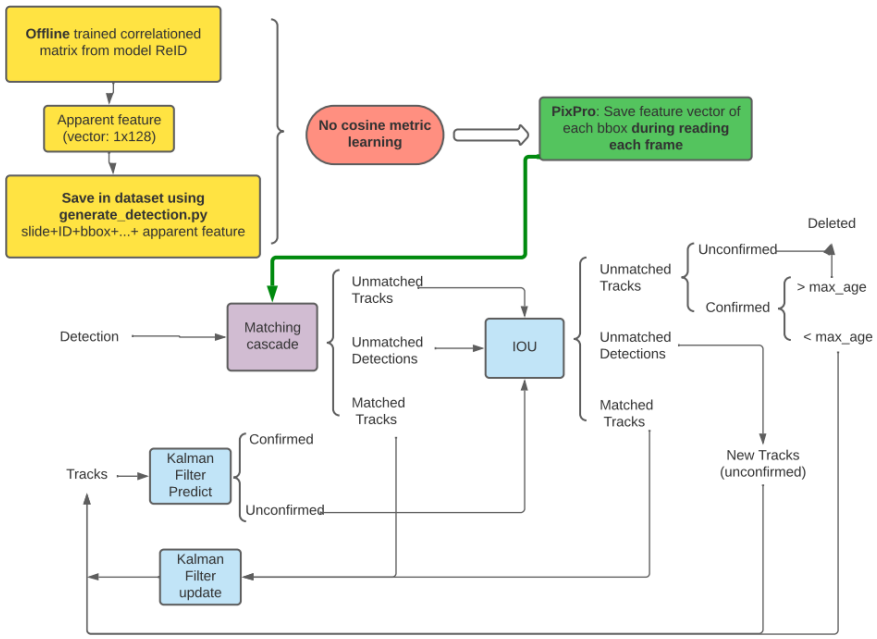

Figure 20: Flowchart of DeepSORT integrated with PixPro model. In the boxes filled with yellow color the method of cosine metrics learning in the original DeepSORT model is described. The box filled with green color presents the modified model, in which the PixPro is integrated in the core algorithm "Matching Cascade" (filled with purple color).

5 Results

The tracking algorithms PixProSORT (Sec. 4.2), PixProSORT with cached features (Sec. 4.3) and DeepSORT with PixPro (Sec. 4.4) were tested alongside plain SORT [**3**] as a baseline, using 2 different scenes; one from the MOT15 [**6**] dataset and one simulated in Blender (Sec. 3.3.2). The evaluation of the trackers are available as HOTA [**10**] metric and sub-metrics.

HOTA sub-metrics provided in this section are; detection recall, detection precision, association recall, association precision. Detection recall is how many of the detected objects are correct whereas detection precision is how many of the objects are detected. Association recall can be regarded as the tendency of the model to correctly predict the trajectories as a whole, and association precision is the tendency to correctly predict shorter trajectories separate from each other. All these values are calculated against a localization parameter α. With a high α value the detections need to be spatially closer to the objects to be regarded as a true prediction, and vice versa. A localization sub-metric is provided separately, which is the spatial closeness of true positives to the ground truth data. So it is expected to be anti-correlated to alpha, whose increase decreases the number of true positives. All these metrics are then combined into $HOTA_\alpha$ metric. To get the final HOTA score, this value is integrated over the range of α. The summary of the metrics can be seen in figure 21.

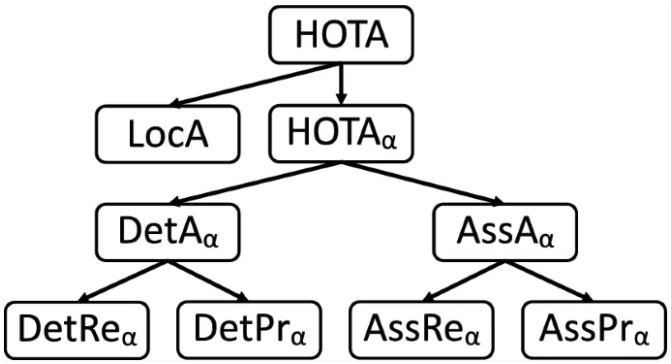

Figure 21: HOTA and sub-metrics[**10**]

5.1 MOT15 Scene

For this experiment, one scene in particular (ETH-Bahnhof) from the MOT15 dataset was used. Figure 22 shows the results of the trackers SORT, PixProSORT, PixProSORT with cached features and DeepSORT with PixPro respectively.

For this scene, the plain SORT algorithm proved to be better than the other algorithms surpassing them in HOTA and all of its sub-metrics. The second-best performer was the DeepSORT algorithm with PixPro. A more interesting comparison for this scene is between PixProSORT and PixProSORT with cached features. By caching the features, the algorithm successfully achieved increasing association recall metric, i.e. its tendency to split a single trajectory to multiple sub-trajectories has decreased. But this behavior also resulted in falsely combining trajectories of different objects into one single trajectory, resulting in a decrease in association precision metric. Overall, the HOTA metric was not significantly affected by the caching method for this particular scene.

5.2 Simulated Scene

In this experiment, the trackers SORT, PixProSORT, PixProSORT with cached features and DeepSORT with PixPro were used to track objects in a simulated scene with Blender, where the objects are often occluded. The evaluation metrics can be seen in Figure 23 respectively.

For the simulated scene, the PixProSORT was the worst performer. But different from the MOT15 scene, PixProSORT with cached features performed significantly better than the other methods. The takeaway is that, since the objects in the simulated scene were often occluded, the re-identification introduced by the cache became an important factor, so that the overall performance increased despite the drawback of the PixPro algorithm. DeepSORT with PixPro surpassed plain SORT in this benchmark, by having higher recall scores. As expected, the matching cascade algorithm of DeepSORT proved to be beneficial in an occluded scene, albeit not as much as caching the feature vectors, providing a middle ground between SORT and PixProSORT with caching.

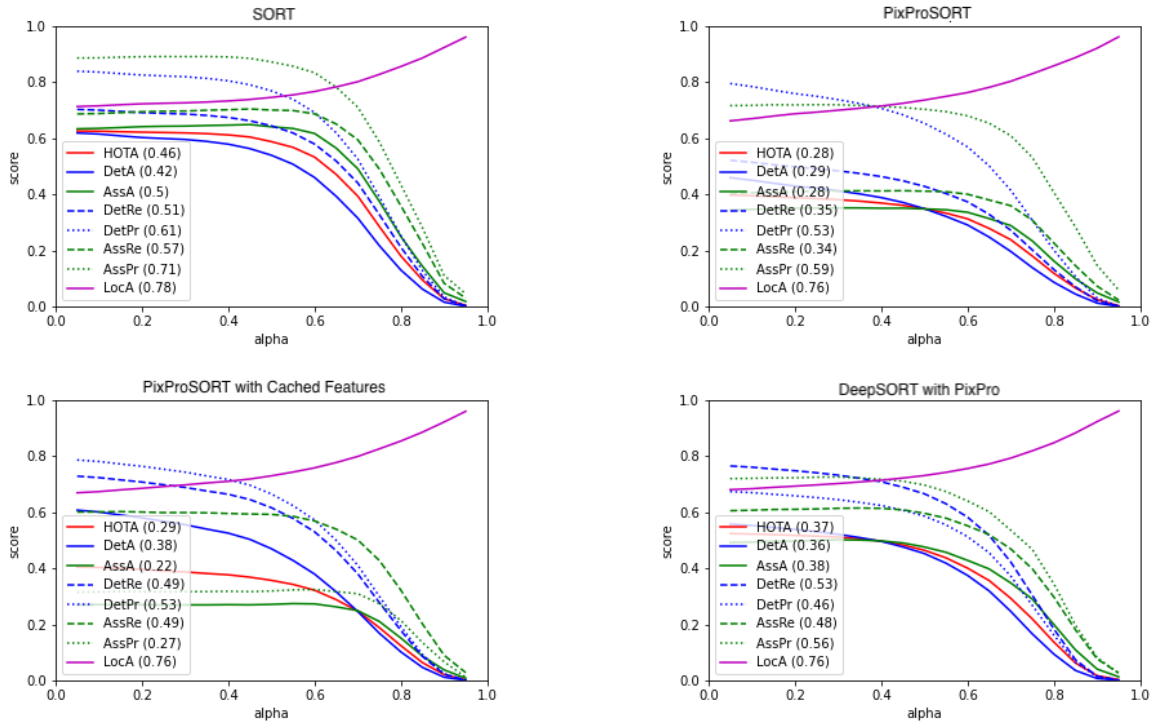

Figure 22: HOTA metrics on the MOT15 scene using SORT (a), PixProSORT (b), PixProSORT with cached features (c) and DeepSORT with PixPro (d) as tracking algorithms respectively.

6 Conclusions

In this project, we analyze the advantages and disadvantages of SORT. We also investigate PixPro to understand how we can merge the visual representations PixPro offers with SORT. We propose different modifications of SORT and compare them with the baseline for the failure cases such as object occlusion and incorrect re-identification of the objects. In addition, we simulate randomized scenes that include the aforementioned modes of failure by using Blender.

PixPro is the backbone of all three modified algorithms since it enables them to extract the visual representations, which is crucial for obtaining appearance information. While PixProSORT enables SORT to receive this information, PixProSORT with cached feature vectors enables PixProSORT to save the feature vectors extracted in the previous frames to re-identify an object. Finally, PixPro integration with DeepSORT modifies the algorithm to use feature representations without generating deep association metrics utilized for appearance information.

The tests on the modified algorithms have shown promising results. PixProSORT with cached feature vectors performs better on object re-identification than the other two counterparts. There are several drawbacks to the proposed algorithms, as on MOT15 scenes, the vanilla SORT algorithm performed better than them. However DeepSORT with PixPro algorithm proved to be a good middle ground between SORT and PixProSORT,

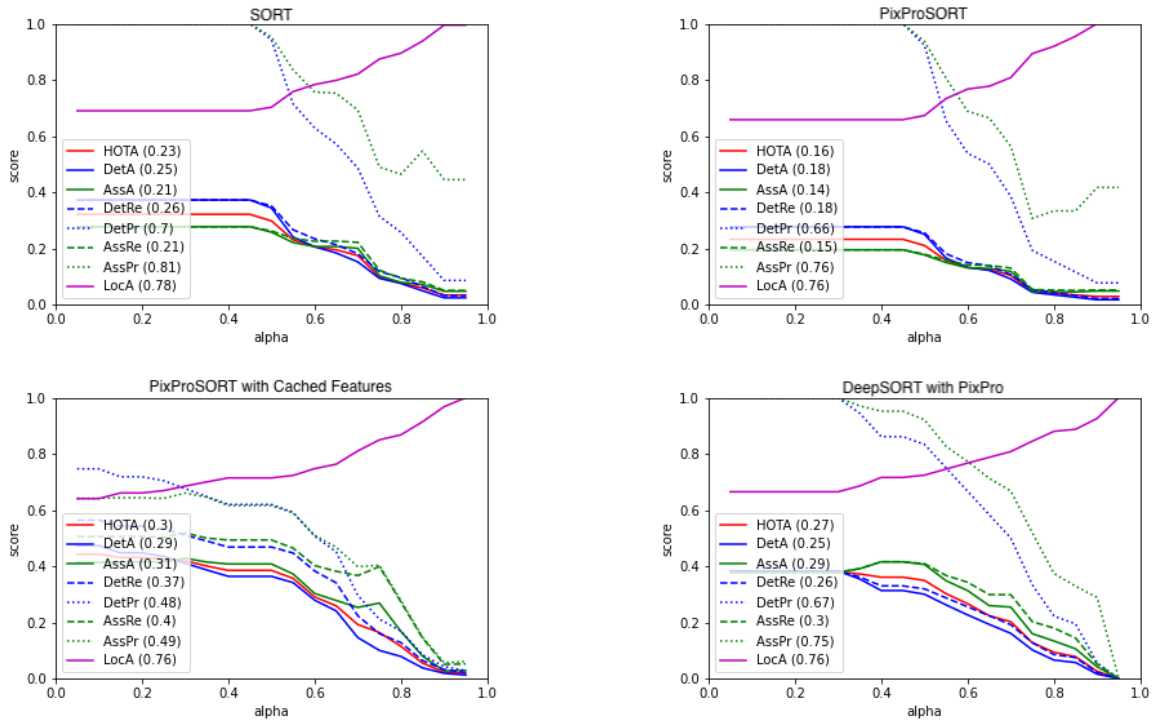

Figure 23: HOTA metrics on the simulated scene using SORT (a), PixProSORT (b), PixProSORT with cached features (c) and DeepSORT with PixPro (d) as tracking algorithms respectively.

coming in second place in both experiments. Also, we assume that some of the bad scores caused by PixPro occurred due to the model being pretrained on an unrelated dataset to the ones used in the experiments.

Taking these outcomes into consideration, several steps might be applied to further the research. First, finetuning on a newly created dataset could be applied. The dataset used to train PixPro is ImageNet [**17**], meaning that although it is familiar to general objects, it is not specifically for food tracking related ones. Thus, finetuning PixPro model with a newly created food tracking dataset could improve the results. In addition, the simulation dataset could also be expanded to cover more modes of failure. Finally, other evaluation metrics, applicable to object tracking, could be investigated.

Bibliography

[1] Gioele Ciaparrone, Francisco Luque SÃ¡nchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, Francisco Herrera, *Deep Learning in Video Multi-Object Tracking: A Survey*, https://arxiv.org/abs/1907.12740, Accessed: 14.02.2022

[2] Yilmaz, Alper, Omar Javed, and Mubarak Shah. *Object tracking: A survey.* Acm computing surveys (CSUR) 38, no. 4 (2006): 13-es.

[3] Deng, Gan, Tao Lu, Emre Turkay, Aniruddha Gokhale, Douglas C. Schmidt, and Andrey Nechypurenko. *Model driven development of inventory tracking system.* In Proceedings of the oopsla 2003 workshop on domain-specific modeling languages. 2003.

[4] Patino, Luis, Tom Cane, Alain Vallee, and James Ferryman. *Pets 2016: Dataset and challenge.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 1-8. 2016.

[5] Geiger, Andreas, Philip Lenz, Christoph Stiller, and Raquel Urtasun. *Vision meets robotics: The kitti dataset.* The International Journal of Robotics Research 32, no. 11 (2013): 1231-1237.

[6] Li, Xi, Weiming Hu, Chunhua Shen, Zhongfei Zhang, Anthony Dick, and Anton Van Den Hengel. *A survey of appearance models in visual object tracking.* ACM transactions on Intelligent Systems and Technology (TIST) 4, no. 4 (2013): 1-48.

[7] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft *SIMPLE ONLINE AND REALTIME TRACKING*, https://arxiv.org/abs/1602.00763, Accessed: 05.02.2022

[8] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun *R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, https://arxiv.org/abs/1506.01497, Accessed: 14.02.2022

[9] Laura Leal-Taixe, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler *MOTChallenge 2015, Towards a Benchmark for Multi-Target Tracking*, https://arxiv.org/abs/2011.10043, Accessed: 12.02.2022

[10] Luiten, Jonathon, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixe, Bastian Leibe *Hota: A higher order metric for evaluating multi-object tracking.*, https://arxiv.org/abs/2009.07736, Accessed: 14.02.2022

[11] Blender developers *Cycles Z-Depth does not conform to the standard Z-Depth output used in other renders*, https://devtalk.blender.org/t/cycles-z-depth-does-not-conform-to-the-standard-z-depth-output-used-in-other-renders/13081, Accessed: 12.02.2022

[12] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, Ross Girshick, *Momentum con-*

trast for unsupervised visual representation learning, https://arxiv.org/abs/1911.05722, Accessed: 14.02.2022

[13] Nicolai Wojke, Alex Bewley, Dietrich Paulus *Repository DeepSORT*,https://github.com/nwojke/deep_sort, Accessed: 12.02.2022

[14] Nicolai Wojke, Alex Bewley, Dietrich Paulus *Cosine metric learning*,https://github.com/nwojke/cosine_metric_learning, Accessed: 12.02.2022

[15] MOT Challenge *MOT Challenge Datasets*, https://motchallenge.net/data/MOT15/, Accessed: 14.02.2022

[16] Zhenda Xie, Yutong Lin, Zheng Zhang, Yue Cao, Stephen Lin, Han Hu *Propagate Yourself: Exploring Pixel-Level Consistency for Unsupervised Visual Representation Learning*, https://arxiv.org/abs/2011.10043, Accessed: 05.02.2022

[17] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. *Imagenet: A large-scale hierarchical image database.* In 2009 IEEE conference on computer vision and pattern recognition, pp. 248-255. Ieee, 2009.

Appendix

6.1 Appendix A: Occlusion with A Familiar Object

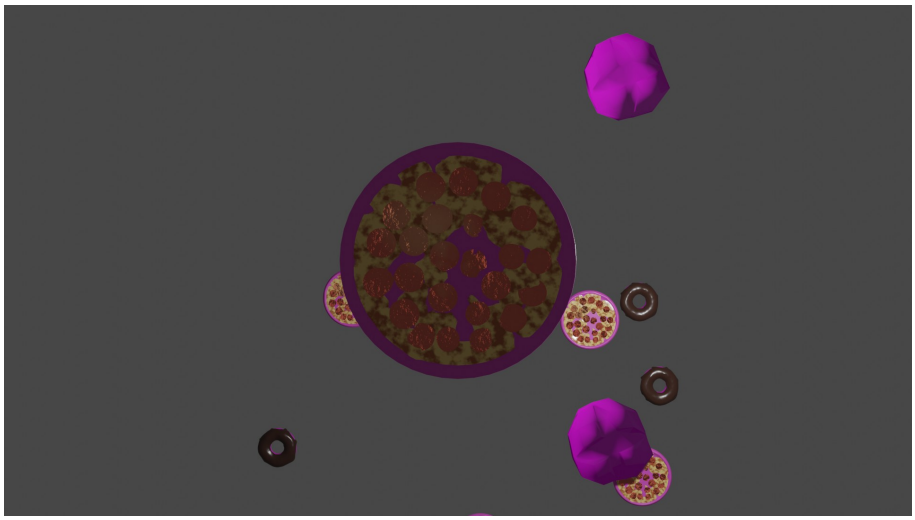

Figure 24: Partial Occluding Pizza Object With Familiar Objects

6.2 Appendix B: Occlusion with A Foreign Object

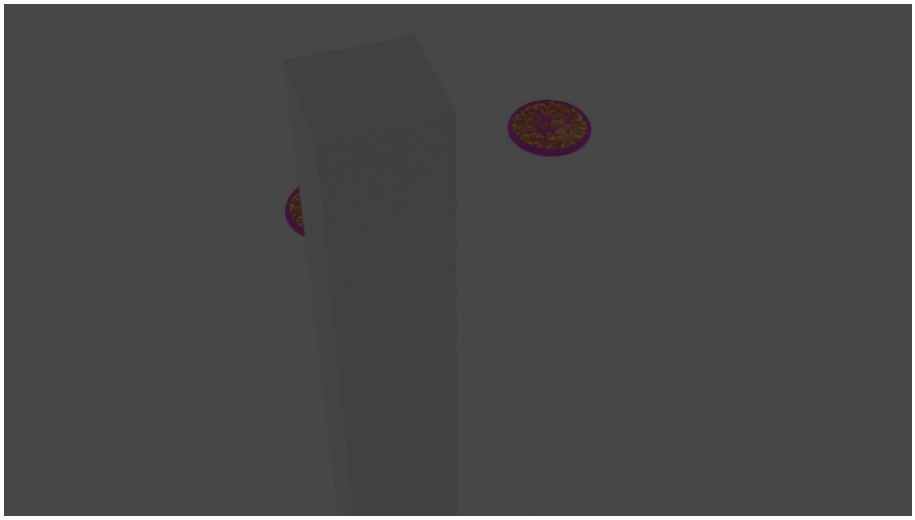

Figure 25: Partial Occluding Cube Object With Familiar Objects

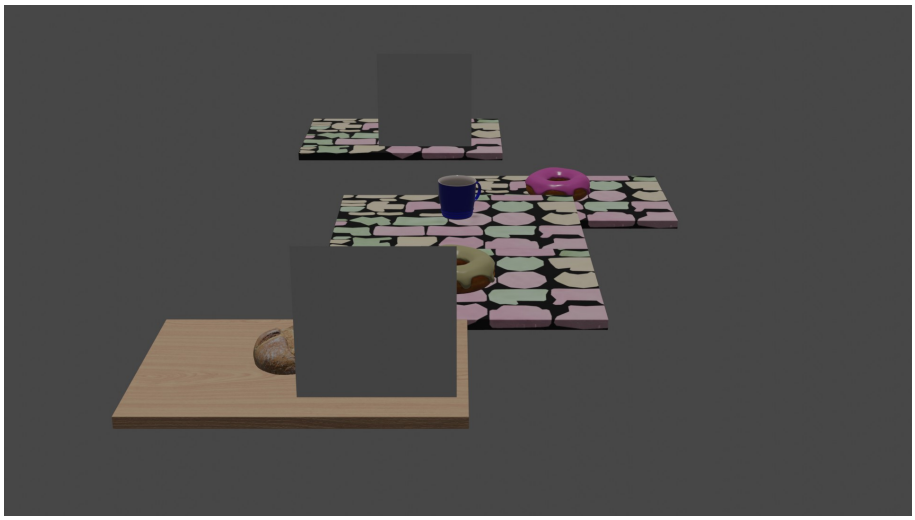

Figure 26: Partial Occluding Palette Object With Familiar Objects