Using Graphs to identify Object Trajectory Clusters

Introduction

To optimise the way Object Tracking is done in current DL methods in terms of Computation cost and time taken, I would like to propose a novel graph-based approach to the Object tracking pipeline. Object Tracking consists of two parts: 1. Detecting objects of interest in the frame and 2. Association/ReIdentification of objects into their past trajectories/IDs. This problem is huge and to tackle the problem in a manageable way, I introduce some **constraints** in the problem. I only work with human tracking and only with videos that come from a fixed frame source like a CCTV footage (instead of a moving frame like an autonomous car's footage).

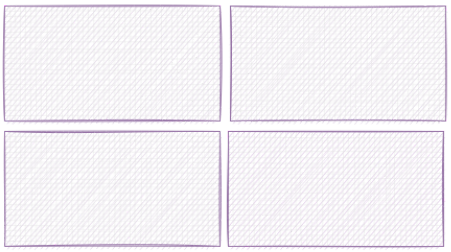
Problem

The DL method of object Re Identification involves passing the cropped image of the agent through a Deep Network and obtaining an n-dimensional **Feature Vector**. We then **pairwise-compare** this feature vector with feature vectors from all the tracks that have been maintained from previous frames using **cosine distance metric** between the two feature vectors. We assign the agent detection to the track that is the most similar above a threshold, else we create a new track. This pairwise comparison can sometimes be time consuming and inefficient in many real-world cases. A person walking in the bottom-left corner shouldn't be compared to a detection in the top-right corner in the next frame. Some other techniques use kalman filtering to predict the future position of an agent and include this score in a final similarity score along with the feature vector similarity score discussed above. This approach does use spatio-temporal information but it still computes pairwise scores for all tracks and the agent prior information (used in kalman filter) is computed for each track(local) i.e it has no global context of the frame as a whole and does not utilise past trajectory data that might have had a similar path.

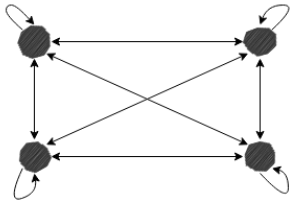
Method

The idea is to divide the frame into n dynamic segments, each representing a node in the corresponding graph. The weighted directional edges going out of a node(including self loop) denote the probability of an object moving from the current node to the neighbouring node in the next time step (not necessarily the next time frame). The dynamic partition of the frame into n constituent parts can be done based on the activity in each partition i.e as the partition becomes more active, we further sub-divide the partition to learn more granular movement patterns in its constituent partitions. This partitioned frame and its corresponding graph generated can then be used to search for a particular detection's associated track. Since the weighted of each outgoing edge is a probability, the sum of all weights of outgoing edges are normalised to sum to 1. Each partition in the frame is connected to each of its neighbouring partitions it shares a side/corner with by creating a non-zero weighted edge in the corresponding graph.

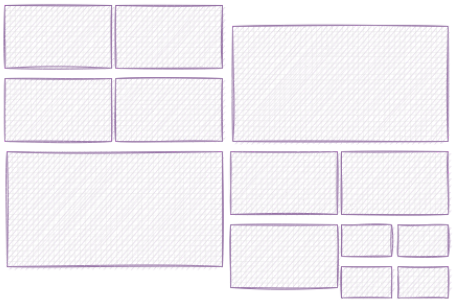
Image Frame



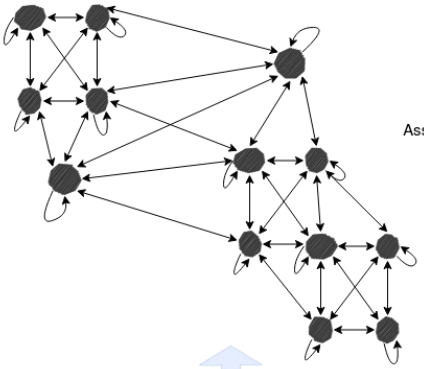
Partition



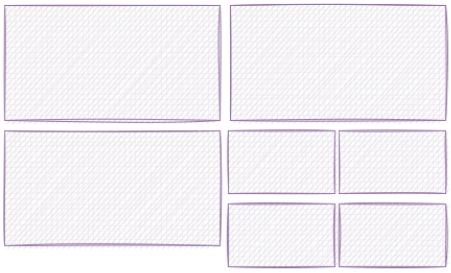
Associated Graph



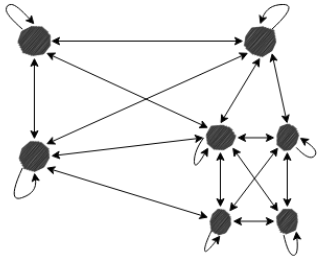
Partition



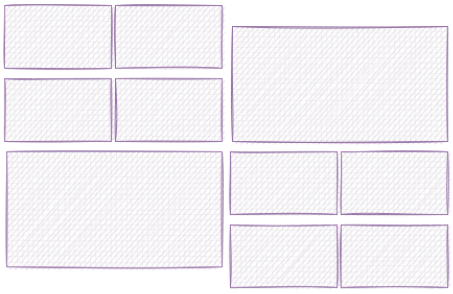
Associated Graph



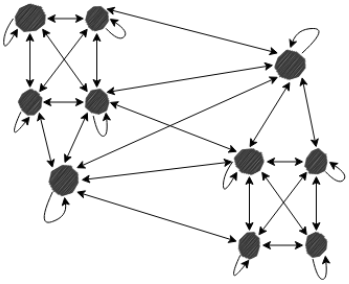
Partition



Associated Graph

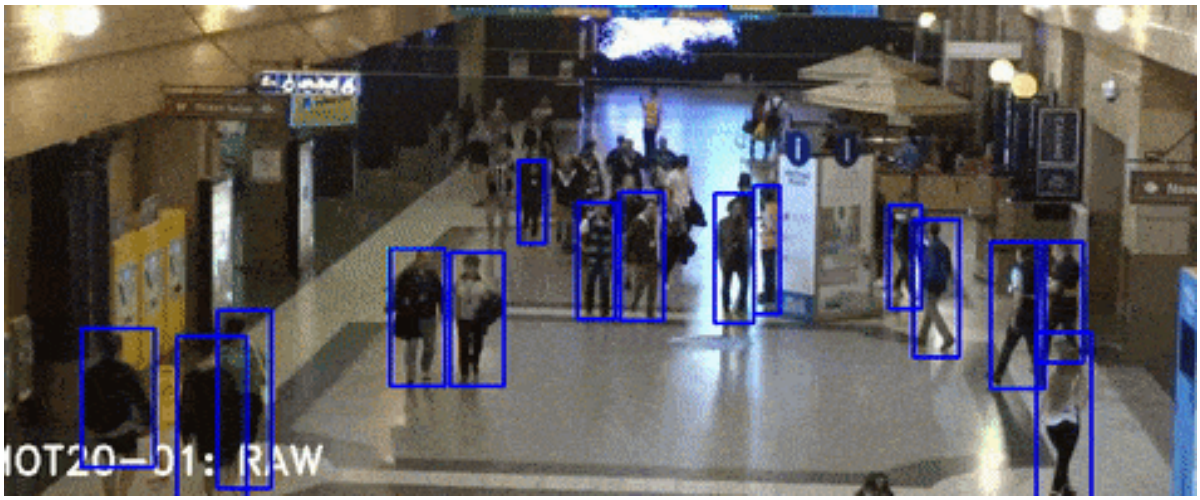


Partition



Associated Graph

We first start off with 4 partitions and a corresponding connected graph with 4 nodes and each edge weighted equally (=>0.25). After some frames, we iterate through all tracks accumulated until now and update weights of edges between the nodes. We now check if the self-loop has a weight more than than a threshold (say 0.5) . This means that for all detections in that particular partition(node), agents are more than 50% likely to stay in the same partition in the next time step. To understand this pattern in a more fine-grained manner, we split this partition into 4 sub-partitions and each of these sub-partition is then connected to its neighbours as shown above. The old node is removed from the current graph and weights of the new nodes' edges are calculated. We do this recursively until all self-loops are weighted lesser than the threshold. This means that we now have a dynamic partition of the frame with granular partitions where it is most busy. We need not perform this graph update operation every frame. Once every few seconds should be enough. And once we do a few iterations of this graph update, the graph should be pretty stable for the rest of the lifetime of the camera as CCTVs typically have a fixed number of patterns like below.



The way we use this graph is that whenever we have a new detection, we find which node it belongs to. We then find the node that has the highest probability of coming into the current detection's node by looking at the incoming edge weights. We then get all the existing tracks that were last seen in this previous node and run a DL based cosine similarity between the detection and the track using appearance based feature vectors. If the detection doesn't match we move to the next highest incoming weight until we finish looking for the associated track in all of the node's neighbours. If still not found, we search for the neighbour's neighbours i.e we perform a BFS-type search in all nodes. If none of them match the appearance satisfactorily, we generate a new track. This process helps us run the re-identification process in a logical way and stop searching as soon as we find a good match instead of running a pairwise comparison of all detections with all tracks.

Additional Improvements / Future Work:

1. Implement an ensemble of graphs to detect and perform well on changing patterns Eg: Trajectory patterns may be different during the Days v/s Nights or during different times of day/ different groups of people.
2. Find partitions that have little to no activity. Can be done using a different graph with same nodes as the one described but with weights of edges quantising the frequency of movement from one partition to another. Thus being able to run detectors only on the frame areas with activity, thereby decreasing infer time and increasing performance in the important areas. Could run the detector on the whole frame once every few frames