Real-Time Identity Tracking via Adversarial Networks

Taraka Pranav Rayudu
Department of Computer Science
The University of Texas at Austin
Austin, Texas
tarakapranav@gmail.com

Shristi Chitlangia
Department of Computer Science
The University of Texas at Austin
Austin, Texas
shristichitlangia2001@gmail.com

Ajith Kemisetti
Department of Computer Science
The University of Texas at Austin
Austin, Texas
kemisettia@gmail.com

***Abstract*—In various applications of Human-Robot Interactions, from a robot aiding with search and rescue operations, to healthcare and social interactions, an autonomous robot will need to follow its human companion in order to perform its respective operations. In the past, many robotics applications have been designed to recognize people in a camera frame. In a static view-frame, there have been successful implementations of identifying people using methods like YOLO. However, tracking people in a dynamic or moving view-frame has continued to remain a challenge. This is due to factors such as people occluding each other when walking, lighting and angle of a given person, and camera angle in capturing the movement of people. In this paper, we seek to further enhance the ability of a robot to track people in real-time with good performance.**

We implemented the algorithm known as You-Only-Look-Once (YOLO) to obtain bounding boxes around a person in a frame of a video. We then went on to implement a tracking algorithm known as DeepSort to further track people in a dynamic frame. We found that our approach works better in situations where the camera angle is as close to a frontal view of the movement of people as possible. In cases of occlusion, we obtained reasonably successful results. However, detecting people quickly and reliably in areas of high occlusion and population density continues to remain a challenge.

I. Introduction

While computer models and robots can generally identify various objects and track them in real-time with relative accuracy, most models cannot do the same for people. Object-recognition techniques using You Only Look Once (YOLO) contribute to the manipulation and movement of objects. However, in the context of people, for robots it continues to remain a difficult task of attaching a unique identity to each human body in either a static or dynamic view-frame. An evident problem with this is the complexity associated with the processing of real-time image frames.

In the case of occlusion, lighting differences, or moving bodies in a given data set, the robot is unable to maintain the initial bounding boxes (coordinate references to an object). As a result it loses track of it as the frames progress until it is re-detected. This makes it difficult for robots to consistently track people in a view-frame and makes future applications of following people around a tricky area of research.

As increasing investment and focus towards the ability for robots to work with humans continues to grow, a robot being able to follow a human companion has become paramount. The application can be seen in search and rescue operations, social interactions, and health functionalities. However, the first step in achieving this is to successfully track people in a real-time dynamic view-frame setting. Thus, this paper seeks to further enhance the robot's functionality to track identities in real-time.

This paper begins to explore the use of YOLO, an application of modified Convolutional Neural Networks (CNN), to first create bounding boxes on people. Afterwards, the DeepSORT algorithm is implemented to track them over every frame. To test and research the algorithms, we make use of the Multiple Object Tracking Benchmark System (MOT) that has data sets of people walking in a view-frame. The objective is for the robot to successfully track each of the individuals in real-time. Our intentions are to make the bounding boxes around entities more consistent and prevent them from flickering as frequently. We intend to build on previous work in this paper and use these studies as motivation for this next step.

There has been research focused on having robots assign human identity to tracked faces and bodies using the camera view and implementing the Hungarian algorithm. Other approaches simultaneously track a time-varying number of people in three-dimension and perform visual servoing (VS) [1]. This system of tracking and VS enables the robot to track several people in a room while compensating for large movements and also while visually controlling the robot to keep a selected person of interest within the field of view. This kind of method approaches a more Bayesian probabilistic model of formulation and uses statistical modeling.

A pressing challenge that comes with identity tracking is the occlusion of a person behind another person, which often causes an identity switch. DeepSORT provides an interesting twist on the traditional Simple Online Realtime Tracking (SORT) approach for tracking people. Essentially, it integrates appearance information which in turn incorporates a deep association metric. The paper adopts a single-hypothesis tracking methodology, which is more beneficial for this project as it is cleaner and keeps the camera more focused. [5].

II. Background

A. YOLOv3

In this paper, we use a pre-trained YOLOv3 (You Only Look Once) architecture to generate a set of bounding boxes from a single frame. Version 3 contains incremental improvements over the original architecture [3]. YOLO is a single convolutional neural network (CNN) that uses features from the entire image when making predictions. This allows YOLO to maintain high-generalizability and remain flexible enough to be applied for new problem spaces such as this one. Moreover, YOLO can be expressed entirely as a single CNN which removes the need for a complex pipeline.

To generate its detections, YOLO first divides each image into an NxN grid of cells where each cell is responsible for predicting bounding boxes and reflect a confidence score for those boxes. The cell also predicts a regression for the set of classes that the bounding boxes can represent. Each of the bounding box regressions contain (x, y, w, h, conf) where x and y represent the center of the bounding box with respect to the grid cell, and w and h (width and height respectively) represent the dimensions of the image in relation to the entire image. Conf is the confidence that the box is accurate. By multiplying the class regressions with the confidence for the bounding box, we get a list of confidence scores of classes for that cell. The final bounding boxes are obtained after filtering through all the proposed bounding boxes using a Non-Maximal Suppression (NMS) algorithm described below.

1) Sort proposed detections by confidence score and place into proposed list
2) Remove the highest proposed detection (D) and place into final list
3) Loop through remaining proposed detection (B)
4) Remove B from proposed list if $IOU(D, B) \geq threshold$
5) Repeat from 2 until proposed detections is empty

NOTE: IOU stands for Intersection over Union

The implementation of YOLO leads to some inherent limitations. Since YOLO limits the possible predictions to just B bounding boxes per cell, cases where each cell has more than the correct predictions of bounding boxes results in misses. When detecting multiple relatively small objects in high density (ex. pedestrians at cross-walk), this effect is exacerbated and presents a significant problem.

Regardless, given that YOLO is a single CNN, it remains one of the fastest and most flexible architectures with relatively high accuracy in comparison to other architectures like RCNN or Fast-RCNN [2]. This makes it a good solution for predicting bounding boxes for our use-case.

B. DeepSORT

DeepSORT is a deep association metric that allows us to assign identities to predetermined detections/bounding boxes. The approach can be broken down into two components: Kalman filtering and frame-by-frame data association. The main purpose of the recursive Kalman filter is to reduce noise in the images and produce more accurate representations based on a joint probability distribution of the given variables. In this scenario, the Kalman filter is given the bounding box centroids (the coordinates of the center of the bounding box with respect to the global frame), the height of the bounding box, the aspect ratio, and an optional feature vector describing the features of the detection. For this study, since the feature vector needs to be run through an encoder which might degrade performance significantly, we leave it out. Velocity is held constant for simplicity's sake. It performs a Gaussian distribution to model future positions for that detection. Then, it performs a data association described below.

The Kalman filter produces a useful distribution that new detections can be compared to. These detections need certain metrics to accurately be placed next to the existing distribution. For motion, the chosen metric is Mahalanobis distance, and to handle occlusion, a cosine metric is also used. A final matching cascade algorithm is implemented to ensure that a person who has been occluded for extended periods of time is not left out of the picture.

Since these metrics aren't learned and are more or less only useful online, a separate, offline, discriminator is needed to figure out where people are. For this, the paper uses YOLO as described above. [5]

III. Methodology

DeepSORT and YOLOv3 were used together via a python script we wrote called controller.py. This script effectively fed the output of YOLOv3 as an input for DeepSORT. YOLOv3 returned bounding boxes around all of the identities in the frame of a video, which DeepSORT needed to perform its statistical modeling, primarily its Kalman Filter. controller.py was our main contribution towards extending previous work on this topic.

The functionality of controller.py can be divided into 5 steps: parse user-given arguments, initialize the DeepSORT tracker, initialize the YOLOv3 DarkNet model, define a frame callback method, and process every frame using this method. Each step makes a call to either library, with the end goal of using the two together to track in real-time.

A. Parsing user arguments

Both Yolov3 and DeepSORT require certain user inputs that are used to parameterize their functionality. A significant component to the input is an input folder that stores the necessary data, which makes it possible for the two systems to work together and track. Among these arguments, only the directory of the MOT sequence is required. This is because the script will later gather the sequence information from that folder and output a live video feed of the real-time tracking. Since every other parameter has a default value that is only relevant to that particular library, only the sequence directory argument will be discussed here.

B. Initializing DeepSORT

After parsing the user arguments, the script we wrote then takes steps to initialize the DeepSORT tracker. It first preprocesses all of the information it finds within the sequence directory. This directory, in our case, will be the folder with the MOT16 image filenames, detections, and ground truths. These ground truths are critical for evaluation of our results. The tracker then initializes its distance metric. This metric can help quantify the distance between a prediction and a detection, which would help to recognize the correct prediction for a detection. The tracker here used a Nearest Neighbor metric as a policy for selecting the right prediction. On a high level, this policy lets a detection's nearest neighbors vote on its prediction. The prediction with the highest number of votes is the prediction that will get assigned to that detection. The mechanism for deciding what the nearest neighbors is determined through a cosine distance. This distance leverages the dot products of two vectors in order to understand their proximity to each other as follows:

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{te}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \quad (1)$$

The formula above is a normalized sum of the products of all the corresponding components of the two vectors. Such a mechanism contributes the detections that constitute the current one's nearest neighbor set.

C. Initializing YOLOv3

Initializing the YOLOv3 model was not quite as complicated as initializing the DeepSORT tracker. This simply required initializing the DarkNet model (the Neural Network architecture used by YOLO) with the input image size. This was 416 in our case. Afterward, we simply loaded the model with the pre-trained weights we downloaded and set it to evaluate instead of train.

D. Defining a frame callback method

The purpose of defining a frame callback method is to put every frame in a video feed through both the YOLO model and DeepSORT tracker. Without this method, only one of the two libraries could be used per frame.

Reading the frames included a call to OpenCV to read the frame in color. The next step was to pass this frame to the YOLOv3 model. The latter took the form of a Tensor variable in order to suffice the parameter of a PyTorch model (included simple conversion of the image to a Tensor variable).

The key observation that enabled the ultimate connection was recognizing the output of the model to also be a tensor. Tensors, in general, are convenient for neural network computations, but not for reading. This prompted a conversion of the tensor to a numpy array, which proved to be an easier solution. By iterating through the array, we retrieved tuples of bounding boxes with information regarding bounding box locations, class predications, and class confidences. Since the DeepSORT tracker required detection objects to run, we had to manipulate the data derived from these tuples. A detection object requires the top left x,y coordinates of the bounding box, width, height, object confidence, class score, and class prediction. The confidence is the percent certainty of the class prediction.

E. Process every frame

With the frame callback method written, processing the entire video was as simple as calling the method for every frame with one caveat: writing results to a text file. The purpose of this text file is to understand the contributions made by our approach, mainly through comparison to YOLO's ground truths. The information to write out was rather simple: frame index, frame id, bounding box top-left coordinates, width, and height.

IV. Experimental Setup

As described above, our first step in conducting the experiment was implementing YOLOv3 and PyTorch in order to get the initial bounding boxes of people in a static frame. We used the PyTorch-YOLOv3 implementation [4] to log the average number of boxes per frame and compare it against the ground truth. In this case the ground truth is provided by the MOT-16 data sets which record the bounding box predictions and actual coordinates in respect to the frame.

Additionally, using our DeepSORT implementation, we were able to log the average number of identity loses (flickering boxes) that occur throughout the video.

In a combination of these two procedures, our controller.py file implements the two features and is successfully able to track the bounding boxes in real-time while also logging the average frame rate of each sequence of the data set.

The experimentation includes testing on five of the MOT-16 data sets (MOT16-02, MOT16-04, MOT16-05, MOT16-09, and MOT16-10);we run these against our script and compare it to that of the ground truth.

V. Results

The experimentation used three indicators to define a successful trial. The first is stable bounding boxes which is computed by dividing the average number of bounding boxes per frame detected in our script with the average number of bounding boxes per frame from the ground truth. The second indicator is the average frames per second that results from the controller.py script. And finally, we have a flickering number of boxes indicator which is the total number of flickering bounding boxes (detect ions that inconsistently track a given person) through the data stream.

As evident in Figure 1, the MOT16-05 data set had the highest average success rate with 95.64% accuracy. Whereas, the MOT16-02 data set had the lowest average success rate of 28.29% accuracy. The average success rate through all the experiments yielded in a 58.79% accuracy.

In comparing the second indicator of average frames per second, figure 2, MOT16-05 yielded the highest average frames per second with 17.48 frames, in contrast to the MOT16-04 data set that has an average of 9.58 frames per

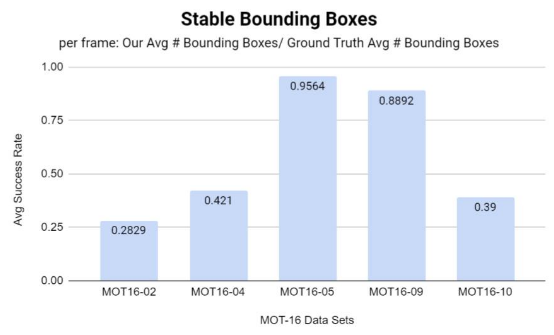

Fig. 1. Stable Bounding Boxes determined by average detection per frame

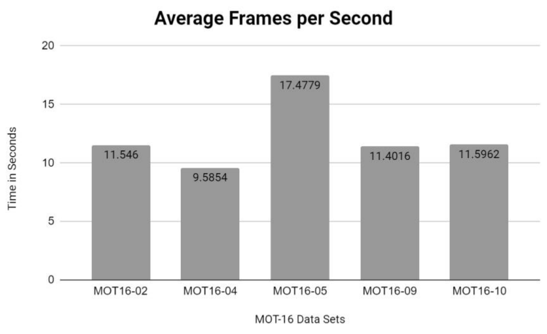

Fig. 2. Average Number of Frames per second

second. The average frames per second through all the trials was 12.32 frames.

The third indicator of flickering boxes is depicted in figure 3, where MOT16-05 experienced the lowest number of flickering boxes throughout the stream at only 3 flickers. In contrast, the MOT16-04 data set experienced 40 flickers in the data stream, showing that our script generally tended to have the worst performance on the MOT16-04. The average number of flickering boxes across the 5 tests was 16.2 flickers per data set. This is relatively quite high considering that the average number of detection made per frame per second was 7.87.

To summarize, in our experimentation the MOT16-05 was the most successful data set based on the three indicators and the MOT16-04 tended to be the worst performance. Differences and inconsistencies can be attributed to YOLOv3-PyTorch's limitation on detecting the number of bounding boxes given a specified cell as per the implementation and as explained in the background. Another factor was that in the case of high population density data streams like that of MOT16-04. In these data sets, there were high levels of occlusion which our algorithm was unable to track most of the time. Specifically in the MOT16-04 data set too, the camera is at a bird-eyes perspective which makes it harder to recognize the people as moving objects to be tracked. In some of the data sets, our algorithm had a lower accuracy when detecting non-moving individuals who were either standing or sitting down.

On another note however, our algorithm was successfully able to track people who were within 100ft of the camera, and in conditions where the movement of people to each other was relatively slower. When there were less people in a given frame, and the camera was placed at a direct frontal view, the data yielded more positive results. As evidenced by the MOT16-05 dataset, we tended to get successful results when conditions better met the YOLOv3-PyTorch implementation.

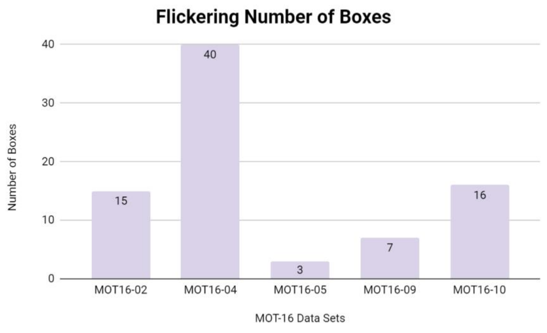

Fig. 3. Inconsistent Bounding Box Detection in the entire data stream

VI. Discussion

Tracking identities in real-time is more accurate in respect to having people within 100ft of the camera, and when the movement of people in relation to each other are not in high density areas. It additionally provides evidence that YOLOv3-PyTorch is not the strongest approach to consistently detect all the people in a given frame and create its respective bounding boxes. In terms of robotics applications, this can be applied for further developments of real-time identity tracking where alternative approaches to YOLOv3 should be attempted. Since this paper applied both YOLOv3 and DeepSORT, other research should further implement the functionalities of DeepSORT to accurately track people. The research conducted in this paper contributes to the ever expanding field of robots tracking people in real-time dynamic view-frames. Eventually, it can be applied to the functionality of human-robot interactions to have robots following people around.

VII. Conclusion

Although there have been previous approaches taken to real-time identity tracking, complexities regarding occlusion, density, and clarity of people in a given frame continue to pose a challenge. In this paper, we further developed the YOLOv3 and DeepSORT models (part of CNN architecture) to better tackle the issue of occlusion and create more accurate detections of people. Although to an extent our research was successful, limitations regarding YOLOv3-PyTorch detection models made it so we had a limited number of bounding boxes tracked using our algorithm. We found the most success in data sets that had direct camera views of the people walking, and such that the people near the camera were within 100ft of the lens. Tracking became harder in conditions of high population densities that made occlusion a significant problem, often switching the identities of the original people being tracked.

This study shows that real-time object tracking is a viable solution for problem spaces with relatively low amount of objects. The findings of this study have important implications for future work. Any future work that needs an identity tracker or seeks to enhance identity tracking should start with this study. This is mainly because of how this work mitigates any flickering bounding boxes that could cause inaccuracies. Additionally, this study also has potential improvements that can be made to it in future work. Future studies could approach this problem with more resources, such as GPUs or supercomputers, and attempt to have tracking use a more parallel computer architecture.

REFERENCES

[1] Yutong Ban, Xavier Alameda-Pineda, Fabien Badeig, Sileye Ba, and Radu Horaud. Tracking a varying number of people with a visually-controlled robotic head. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4144–4151. IEEE, 2017.

[2] Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[5] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.