Pedestrian Detection and Tracking

Abhijit Hebsur, z5220436

Akhil Jain, z5221116

Denzil Saldanha, z5227714

Peter Nguyen, z5061984

Ying Huang, z3234203

Abstract*— This report presents an implementation of Computer Vision detection and tracking algorithms with real life applications in the detection of human in everyday situations with practical applications. The detection and tracking of multiple pedestrians moving individually and in groups within a set area and throughout the multiple frames in real-time pedestrian footage is what the solution provided in the report is seeking to solve. The approach taken was to implement hand-crafted or neural networks and deep learning-based detection solutions in order to identify the pedestrians then using a combination of Kalman Filter, Centroid and Deep Sort solution in order to track the pedestrians after detection throughout the frames individually and in a group. The neural networks and deep-learning based-solution with Faster Region-Convolutional Neural Networks detection with the Centroid algorithm providing the most accurate pedestrian and detection solution. Hence, neural networks provide a better detection solution than handmade features whilst providing a simple tracking algorithm would provide better performance in the final tracking solution.

Keywords—pedestrian detection, tracking, Contour, HOG, Haar Cascade, Faster R-CNN, YOLO, Kalman Filter, Hungarian Algorithm, Centroid, Deep Sort, computer vision.

I. INTRODUCTION

The Group Component of the Final Project has the requirements of detecting and tracking pedestrians throughout multiple frames as well as keep track of how many pedestrians are within a set boundary, and how track how many pedestrians are currently within a group. All the of required are broken down into three fundamental tasks in order to achieve the requirements as per the specifications which are to be completely implemented using Python.

Task 1 requires the solution detect and track all the pedestrian movements within the video image sequence frames. The first specification of Task 1 is to detect all the moving pedestrians and to draw a bounding box around it. The second specification of Task 1 is the draw a trajectory path of each pedestrian to track their location and their movements. The third specification of the task requires to keep a real time counts of the pedestrian that is in the frame since the start of the first video frame.

The successful implementation of Task 2 requires the user to be able to draw a rectangular bounding box on the first frame and subsequently keep track of all of the pedestrian who enter and exit the bounding box and keeping the real-time counts of the pedestrians within the boundaries.

Furthermore, Task 3 requires the solution to be able to detect group behaviour in the image sequence. The first specification is to provide real-time information of how many pedestrians walk in group of more than 1 person and how many walks alone. And finally, for the solution to display a bounding box around a group of pedestrians that stay together for more than one frame and destroys the group detection as soon as the group disperses.

The dataset given is a set of 795 images which represents the image sequence that is indicative of what each of the video frames would be. The dataset provides us multiple frame that are separated from the original output for the solution to detect and track pedestrians from each of the frames using the respective algorithms and experimental methods which will be later discussed in order to provide that out that is specified in the requirements as mentioned previously. These specifications and dataset will allow our solution to detect and track multiple pedestrians in real-time across many frames, count how many pedestrians within a boundary, display a boundary and keep track of how many pedestrians are in group and disperses, and how many are walking alone which will form the basis of the complete solution to the specifications.

II. LITERATURE REVIEW

Multiple object tracking consists of localising and identifying all objects of interest in such a way that their identities are consistent throughout the video. A good model for object tracking not only has to accurately detect and draw trajectory of the objects but will also have to

analyse the number of people forming the groups and leaving them. Ideally, the objects detected should keep their former identification when reappearing in the video and the model should not get confused if the paths are intersected when objects move around. Multiple techniques have been tried to identify the objects such as Frame Differencing, Optical Flow and Background Subtraction.

In our report we have tried both handmade features and deep learning techniques to solve the detection problem. The faster R-CNN have outperformed all the other detections methods. When it comes to prediction of trajectory of the objects, multitude of approaches such as Kalman Filters, RNN, SVM, have been suggested to model the velocity of objects and predict the position of future frames. This problem is essentially a filtering problem where at each step, the goal is to model the state of the previous step given the noise around it. Particle filter is traditionally used to solve this problem however, we have used Kalman Filters to optimally estimate the state of linear dynamic system. The next part to solving the tracking problem after prediction is determining what detection corresponds to which object based on the prediction from the previous step or a new object. New problems arise when the objects leave the frame and reappear as the detector can then produce false positives. The first method used to solve this issue is Hungarian Algorithm which accepts a Cost Matrix C and in four steps, the matrix is manipulated to give the optimal match with time complexity O(n3). However, Similarity Measures resulted in better performance as it computes Intersection Over Union (IOU) between bounding boxes.

Recently, Channel and Spatial mixed Attention CNN has been used for pedestrian detection where channel can associate features and spatial attention module can aggregate similar feature to illuminate pixels. Experiments show that CNN has outperformed other modules.

III. METHOD

A. Detection Model

The report explored and compared the performance of 3 handcrafted feature models and 2 deep learning models.

1) Contour Detection

The goal behind using the contour detection technique is to detect the boundary of the objects in the images, draw a rectangular box around them and also track the centre of the object using those contours. The image is first converted into Greyscale over which background subtraction is applied to distinguish the objects from the background. Once the objects in the image are clear, morphological transformations such as closing, and opening are applied to remove noise from the images. The image is then dilated to enhance the size of foreground objects. The above pre-processing of the image will lead us to much better contours and thus better detection accuracy. The dilated image after pre-processing is then used to find the contours using the in-built OpenCV library function. The next step is to find out the centroid of the object and we will use Image Moments to find those centroid points. Image moment is a certain particular weighted average of the image pixels' intensities. So, we will consider the area inside the contour for finding the image moments [7]. Let the image moment inside the contour be M then the coordinates of the centroid can be calculated as follows:

$$cx = int(M['m10']/M['m00'])$$
$$cy = int(M['m01']/M['m00'])$$

Once the centroid points are obtained, this centroid point will represent the object so we can put a bounding box for the object with respect to the centroid. The dimensions of the bounding box can be obtained from the contour using the function cv2.boundingRect(contour) which will give all the dimensions of the bounding box. The final thing to do will be to draw the bounding box around the dimensions.

2) Histogram of Oriented Gradients

The use of the Histogram of oriented gradients (HOG) feature descriptors computes the gradients in the region that needs to be described which is the placed into bins according to their orientation. Then the cells are grouped into large blocks which are all each normalised where Linear SVM classifiers are trained to determine which part of the image is a pedestrian or not [9].

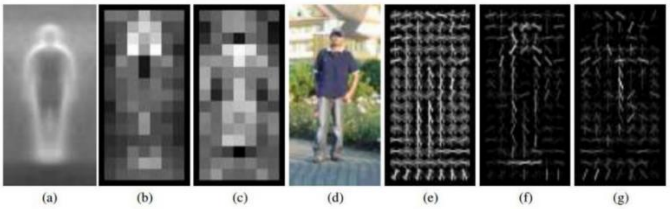

Fig1: Different stages of the HOG detection.

From the test image d), the average gradient image is outputted over the training examples in a) where in b) and c) each pixel shows the maximum positive and negative SVM weight respectively in the block centred on that pixel. The HOG descriptor is displayed in e) where the positively and negatively weighted SVM weights are displayed in f) and g) respectively. The extra overlapping boxes that shows different features of the pedestrian's body is reduced to a single bounding box in the pedestrian detection in the HOG detection solution is implemented via the non-maximum suppression algorithm. This is used

to reduce the false-positive detections in the final detector [6].

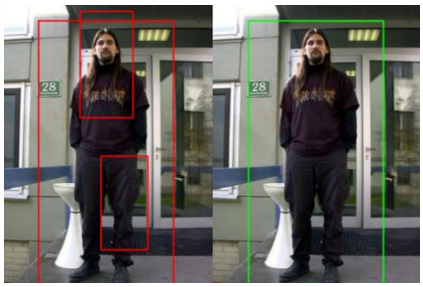

Fig2: On the left HOG features were used to extract the pedestrian's multiple features.
On the right, the overlapping features within the bounding box were suppressed with NMS.

3) Haar Cascade

Object detection using Haar feature based cascade classifiers is an object detection technique that was proposed in the paper "Rapid Object Detection using a boosted cascade of simple features" [11]. This machine learning algorithm is trained on positive and negative samples which is then used to detect other images. The training is done by extracting Haar features. From each features a single value is obtained subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle. There is a good resemblance between the convolutional kernel and Haar like features.

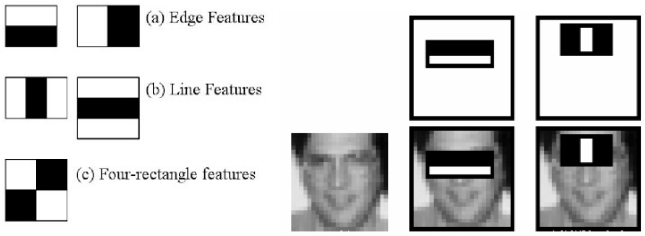

Fig3: On the left Haar Features used to detect the features in the training examples.
On the right good features extracted by HAAR for face detection

The sum of pixels under the white and black rectangles is found. This is done by using integral images which reduces this calculation to an operation involving just four pixels. The number of Haar features calculated is always large and not all the features that are calculated from this are good. Some good features as show in the figure could include the eyes are darker that the nose bridge. These features are extracted using Adaboost. Which basically finds the threshold which classifies the given image into the correct class. Only the features with the least error rate is chosen. This is an iterative process as the weights are updated iteratively. A final classifier is a combination of all weak classifiers [2].

4) Faster R-CNN

R-CNN (Region-based Convolutional Neural Network) consists of 3 simple steps:

a) scan the input image for possible objects using an algorithm called Selective Search, generating ~2000 region proposals
b) run a convolutional neural net (CNN) on top of each of these region proposals.
c) take the output of each CNN and feed it into a) an SVM to classify the region and b) a linear regressor to tighten the bounding box of the object, if such an object exists.

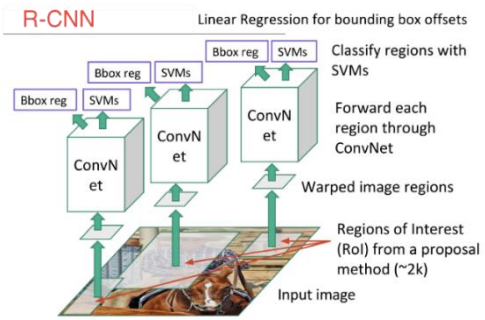

Fig3: R-CNN architecture

Faster R-CNN (Faster Region-Convolutional Neural Network) is the 3rd version of R-CNN sequence, which extracts features and proposes regions of interest in the image and classifying them according to known categories, using CNN as the feature extractor. The idea is the same since its first version (R-CNN), when they have used Selective Search for region proposals [10]. The Faster R-CNN is composed by two main modules:

a) region proposal network (RPN) to efficiently choose relevant regions;
b) neural network that classifies the proposed regions to predict a class and class-specific box.

5) YOLO

YOLO (You Only Look Once) integrates the extraction of the candidate boxes, the feature extraction, the target classification, and the target location into a single deep network. It divides the image into a grid, and for each cell in the grid considers number of possible bounding boxes; then uses neural networks to estimate the confidence that each of those boxes contains an object and class probabilities for this object. The loss function composes of classification loss, the localization loss (errors between the predicted boundary box and the ground truth) and the confidence loss. [5]

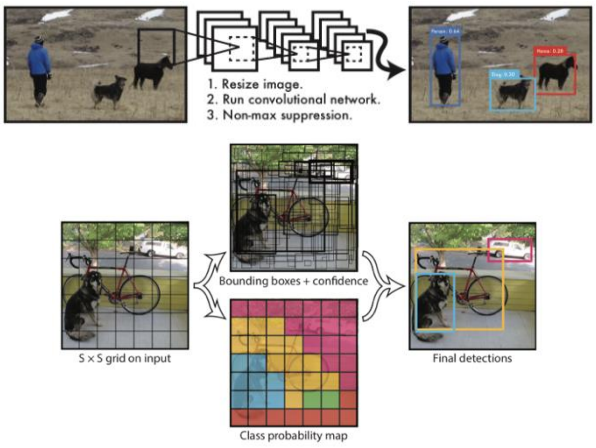

Fig4: YOLO detection architecture

YOLO removes all fully connected layers and uses pre-defined anchor boxes to predict bounding boxes [14]. The sizes and scales of anchor boxes in a single grid were pre-defined (normally 5 anchor boxes), similar to Faster R-CNN (9 anchor boxes) [13].

YOLO has difficulty detecting objects that are small and close to each other since each grid cell only predicts two boxes in a grid predicting. Faster R-CNN on the other hand, do detect small objects well since it has nine anchors in a single grid, however it fails to do real-time detection with its two-step architecture [13].

Compared to YOLOv2, YOLOv3 deploys106 layer convolutional network for detection and 9 anchor boxes for bounding box estimation, which improves accuracy but at the cost of slower speed. YOLOv3 makes prediction at three scale with feature map size at 13x13, 26x26 and 52x52. Fine grained of the feature helps address the issue of detection small objects [3]

B. Tracking Algorithm

1) Kalman Filter and Hungarian Algorithm

Multi object tracking is done by using a method called Kalman Filters which can predict future positions based on current position. Finally, Hungarian algorithm connects all the prediction to tracks and produces the tracking results of multiple objects. Kalman filter is used to predict the new detection in the consecutive frames and to associate those predictions to the location of a track in each frame. Hungarian Algorithm can tell if an object in current frame is the same as the one in previous frame. It will be used for association and id attribution. It is used to create new tracks if no tracks are found. Based on the centroids of the objects a cost matrix decides the cost of the new tracks using Sum of Square Distance between predicted and detected centroids. This algorithm assigns the correct detected measurements to the predicted tracks and will also remove the tracks which have not been detected for a long time. The tracks are then finally used to update the Kalman Filters state and the tracking result is displayed to the user [1].

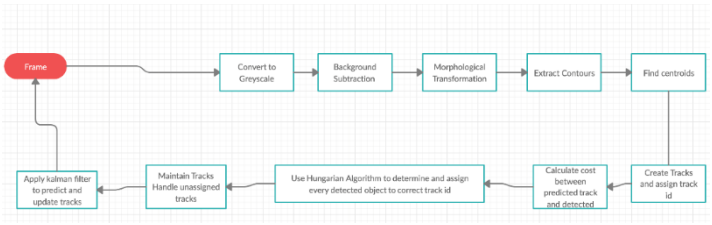

Fig5: Kalman Filter and Hungarian Algorithm tracking flow

2) Centroid Tracking

The centroid algorithm assumes that a list of bounding boxes is passed for every frame. This bounding box represents the (x,y) co-ordinates of the pedestrian. Once the bounding box is presented an id is assigned to it and a centroid is calculated for each bounding box. This is calculated using the diagonals of the bounding box. But for each subsequent frame, when the new bounding boxes are received, they're Euclidean distances are calculated with respect to the old centroids. If the centroid is deemed to

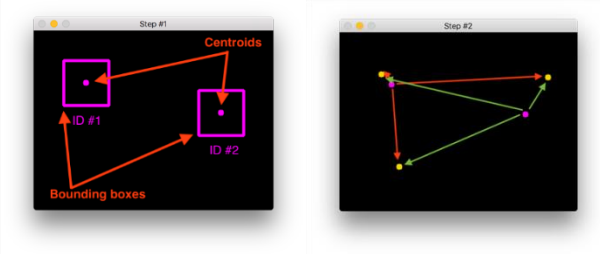

Fig6: an image depicting centroids (left) and bounding boxes (right)

Euclidean distance calculated between the centroids belong to an old id, the (x,y) coordinates of the centroid is updated. We then register new objects that have entered the frame. If a centroid has not been detected for a certain number of frames, it is then destroyed.

Every pedestrian is associated with these centroids and every centroid is associated with the ID. The trails are drawn using the dictionary of deques, with a max buffer length of 10.

3) Deep Sort

In extension of Sort (Simple Online and Realtime Tracking), deep Sort extracts deep learning feature descriptor from a pre-trained model and associates detected objects across different frames based on the coordinates and velocity of detection results as well as the learned features. Obtaining the appearance feature vector

to distinguish each object effectively reduces the number of switching between identities, ensuring that the tracking is more stable and accurate [5]. If there is no relevant box in the next frame, the algorithm assumes that the object has left the frame. The quality of tracking depends on the quality of object detection.

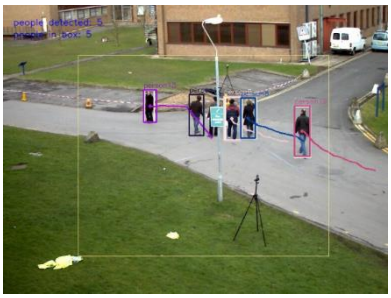

Fig7: Deep sort tracking

IV. EXPERIMENT SETUP

In the execution of the solution in the experiment, the frames of the image sequences are combined to produce a video which will be the input that our solution reads frame by frame using functions from the OpenCV library.

Google Colab is a free cloud service. It allows you to run your python scripts on cloud and provides free GPU processing. Google Colab syncs with your Google Drive for storage and can thus access files that are stored in there. Google Colab also is very useful for collaborating with multiple developers on a particular code. For experimental purposes Jupyter notebooks was used to run the programs.

A. Pedestrian Detection and Tracking

1) Contour and Kalman Filter

All in the images in the sequence were read in a sorted manner and video was formed in the same sorted sequence. From the video, each frame was used to form the contours and determine the centre. The centre points of all the objects present in the frame are further passed to the Kalman Filter to predict the trajectory.

2) Histogram of Oriented Gradients

All the frames were iteratively read into the initialised HOG descriptor with the Linear SVM Classifier to detect the pedestrians where the rectangles were extracted from the function that detects multiple objects before the non-maximum suppression algorithm combine the overlapping bounding boxes of the pedestrian in the image.

3) Haar Cascades

All the frames were iteratively read into the classifier. Each frame was then read into the classifier. A xml file containing the "full body haar cascade" was used to detect the pedestrians in the images.

4) Faster R-CNN and Centroid Tacking

The Faster R-CNN model was implemented with TensorFlow CPU libraries on Google Colab based on "faster_rcnn_inception_v2_coco_2018_01_28" pretrained model from TensorFlow Detection Model Zoo was used for pedestrian detection task in this project. The model was trained on the COCO dataset commonly used for object detection and covers 91 object categories in 2017 releases [4]. The model package has the its variables converted into inline constants so everything's in one file and ready for serving on any platform including mobile. Freezing process includes loading the GraphDef, pull in the values for all the variables from the latest checkpoint file, and then replace each Variable op with a Const that has the numerical data for the weights stored in its attributes It then strips away all the extraneous nodes that aren't used for forward inference [8].

The model achieves a reasonably high accuracy (92%) of pedestrian detection while reasonably fast speed (average of 1.5 seconds per frame). The output boxes from the model for each frame are filtered conditioning on if the class is 'person' and the confidence score is above 70% and box height is above 35 pixels to avoid false classifying road barricade as a person.

5) YOLO and Deep Sort

YOLO v2 and YOLO v3 were deployed for detection via TensorFlow CPU environment. YOLO v3 was implemented on Keras with TensorFlow as backend and pre-trained objection detection model weights [15].

YOLO v2 was much faster to finish detection of the 795 frames while YOLO v3 took on average of 3 mins per frame for detection as a result of more comprehensive detection and classification algorithm [3] and CPU is not the ideal environment for TensorFlow.

The detection result only keeps boxes when classification class is 'person' and ignores other classes, taking default IOU of 0.4 and the confidence score of 0.4, which decides whether the detection box is kept or not. The higher the confidence score, the stricter the detection box is returned in the detection output. The selection of the threshold is a trade-off between false positive (too low) and false negative (too high) detection.

Both YOLO and Faster R-CNN could get confuse other objects as pedestrians for small objects (as figures below) and fails to detect when two objects overlap. To minimise the false detection, one approach is to re-train the model on the sample of underlying frames to add barricade feature to the output model weights to

distinguish from person. Without going through the model training path, a quick fixed was setting minimum height of the detected box as 35 pixels to avoid the model detecting road barricade as pedestrian (before and after adjustment figures).

Fig8: Example of false detection (left) and after box height adjustment (right)

Deep Sort was implemented together with YOLO v3 via Keras, which is widely used algorithm. Deep Sort uses pre-trained metric feature representation which is trained on Market1501 with max_cosine_distance of 0.5 and non-maximum suppression overlap of 0.3 [12].

Fig9: Tracking of person 7 and 8

Person 7 and 8 are correctly identified and tracked across 60 frames from the first frame when they are tracked (Fig9. left), as they are consistently detected throughout the sequence and always stay in the frame. Deep sort can correctly identify and associate pedestrians even in the crowds of people.

Fig10: Tracking of person 1, 2, 3

As can be observed from the tracking result, the model failed to detect person 1 and 3 in Fig10.2 due to overlapping with other objects. When they are detected in the next few frames and re-tracked (Fig10.3), they are treated as new tracking object by the algorithm.

B. Pedestrian in Box

In order to successfully calculate how many pedestrians, enter and exit the box in Task 2, the box is drawn around where the user would like to measure how many people enter and exit the box and keeping the opposite corners of the box. With every frame that is iterated in the video, the centroid that was calculated in the previous tracking algorithm representing each pedestrian's centre point is detected and tracked where the coordinates are kept in a list of centroid tuples. The corner rectangle coordinates of the box and the list of centroid coordinate tuples is inputted into the function that calculates the number of pedestrians in the box. Every centroid coordinate in the list is then iterated over where the count for the pedestrians inside the box is incremented if the centre coordinate tuple values is within the boundaries of the box. The values of the number of pedestrians inside the box is returned.

C. Pedestrian Group Detection

The implementation is the extension of the function in Task 2 that calculates how many pedestrians are inside of the box. However, instead of a set defined box like in Task 2, a bounding box is calculated for each detected pedestrian with a bounding with the threshold around the all centre coordinate points for each pedestrian, that was returned from the centroid tracking algorithm to calculate the adjacent people within the threshold. The function then calculates if centroid distance between pedestrians is less than the threshold (40 pixels selected as it was shown to provide a sufficiently close distance to be classified in a group), they are counted as one group and bounded by a rectangle box, otherwise they are counted as a pedestrian walking alone. Since the group will consist of many overlapping bounding box surrounding each pedestrian that was created with the respective thresholds around each pedestrian, the bounding box around the group pedestrian is defined by the minimum and maximum coordinate of the clusters of boxes around each pedestrian that is grouped together where it is then drawn with an additional buffering margin surrounding the group and returns the coordinates where the box around the box is drawn for the video output. And finally, once distance between the centroid's coordinates are beyond the selected threshold in the frame, the group bounding box is destroyed which is representative of the pedestrians splitting from the group.

E. Evaluation Metrics

The ground truth was used as the metric to calculate the accuracy and the success of the detection of the solution. A ground truth was created by visually inspect

how many people and groups were in each frame. For every frame, the number of pedestrians was compared with how many pedestrians are detected by the solution. Furthermore, the same concepts were applied with detecting how many pedestrians are within a group and how many groups there are with the groups detected by our solution to test the success of not only the detection but also the tracking solution.

V. RESULTS AND DISCUSSION

Comparing the hand-crafted methods with neural networks, the hand-crafted methods were able to provide a detection significantly faster than the neural network (almost instantaneously to the naked eye). The neural networks took a couple of seconds even minutes (depending on the computer specs) to give a prediction. In terms of the accuracy of detection, the neural network solutions performed much better with the high accuracy especially when detecting multiple people who are adjacent to each other and pedestrians far away from the camera. Out of the deep learning solutions, the Faster R-CNN detection method was selected for the application given its fast detection speed and high accuracy.

In terms of tracking, among the three mentioned methodologies, the Deep Sort methodology seemed to be the most accurate in terms of being able to keep a track of the pedestrians' movements and identities. While Deep Sort would be the preferable choice of pedestrians tracking, the combination of both Deep Sort and neural network pedestrian detection was computationally expensive and thus was not taken as the choice of approach. Centroid tracking was taken as the approach to move forward with pedestrians tracking as it was able to give a good result. Centroid tracking formed the basis on which Task 2 and Task 2 were completed.

Pedestrian detection ground truth metrics

Metrics	Value
Number of frames observed	795 frames
Number of pedestrian (Actual)	4873
Number of detected pedestrians	4422
Detection accuracy	92%

Group detection ground truth metrics

Metrics	Value
Number of frames observed	795
Number of groups (Actual)	709
Number of detected groups	520
Detection accuracy	73%

Table 1: Summary of detection performance

The results show that the solution was very effective at accurately detecting pedestrian with a detection accuracy of 92% across all 795 frames observed. Other factors on why pedestrians might not have been detected by the solution is that the pedestrians were obstructed by another object or another pedestrian which would've affected the total counts of pedestrian in that particular frame as seen by the figure below.

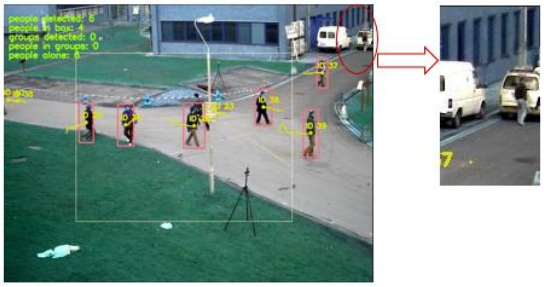

Fig11: Frame of a person blended with another object that wasn't detected in the distance

Other issues are that other pedestrians that is a long distance away enhanced by the fact that they have been blended in with the background objects which must have prevented the pedestrian from being detected by the solution.

Fig12: Frame of a few pedestrians that wasn't detected due to being obstructed

However, despite the lesser accuracy in terms of group detection as compared to individual pedestrian detection, the group detection has a decent amount of accuracy with 73% detection accuracy. However, the less detection accuracy is based on the subjective interpretation of what a group is in each individual frame, in terms of whether each pedestrian is interacting with each other or not. Whereas the tracking and detection solution in determines whether the pedestrian is in a group is purely determined by distance in terms of the threshold and whether pedestrians are within the threshold or not regardless of whether they are interacting or not in a group. The following image is for example according to human observation that the two pedestrians are in a group even though both centroid points of the two pedestrians are not inside the threshold that is considered to be a group. The definition of what a group constitutes is subjectively different to what the solution interprets a group to be.

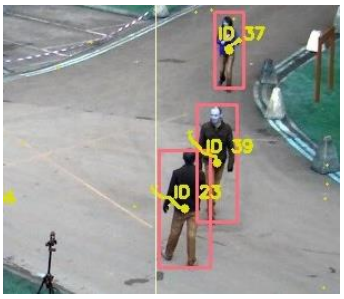

Fig13: Frame of interacting pedestrians in a group that wasn't detected as a group as they are not within the threshold

Algorithm	Frames	Run Time (s)	Second per Frame
Contour and Kalman	795	82.65	0.160
YOLOv2	795	443.59	0.56
YOLOv3	100	16126	161.26
YOLOv3 + Deep Sort	100	23508	235.08
Faster R-CNN + Centroid	795	1168.99	1.47

Table 2: Runtime performance for each algorithmic solution

The hand-made Contour detection features with Kalman has the fastest performance but has the lowest detection accuracy. YOLO v2 is quite fast out of the three deep learning algorithms but at lower detection accuracy. YOLO v3 improves the accuracy from v2 but at a much slower computer speed with TensorFlow on windows. On the other hand, Faster R-CNN on Google Colab runs much quicker and high level of detection accuracy.

VI. CONCLUSION

The performance of our selected model has yielded reasonably good results. Deep learning models outperform handcraft feature detection models in terms of accuracy and the locationality of the bounding box, at the cost of initial environment setup and computation power. Detection accuracy and speed are the two main factors for the model selection. Centroid tracking algorithm is quite intuitive and straightforward to implement and performs quite well on the trajectory drawing. Simple group detection by spatial distance between centroid identifies 72% of groups. And yet further work can be done to achieve better result.

The performance of the algorithm could have been made better by retraining the final layers of the neural network against the pedestrian dataset. The deployed Faster R-CNN model was pretrained on the COCO dataset which is a dataset with multiple classes. Training the last couple of layers of the neural network would specialize the network towards classifying pedestrians to reduce false positive rate and false negative rate. The training dataset could be expanded to include more people in each frame. Currently the dataset has a maximum of 13 people in any given frame. It would also help to improve the prediction of the model under different conditions (indoors/ different weather conditions such as rain etc.).

Hyper parameter tuning could also be done to fine tune the performance of the neural network in order to give out a better result when it comes to the pedestrian dataset.

In terms of model evaluation, a comparison of the model could be done with respect to a standardized ground truth. This would let us know how our model performs in comparison to our models out there. Additionally, other evaluation metrics such as FAF (false alarm per frame), Fragmentation (object is lost in a frame but detected in a later frame) could be used.

The tracking algorithm could be improved by using a more sophisticated method which uses distance, shape and deep appearance measures. Distance and shape measures look only at the position and geometry of the bounding boxes, whereas deep appearance features differentiate the object within each bounding box. Tracking could also be done in real-time by skipping incoming frames until the currently frame is analyzed.

For group detection, more sophisticated algorithm can be implemented, e.g. group level of metrics can be obtained, learned and track to better classify different formation of groups. The current algorithm applies one centroid distance threshold to detect the group with no tracking across frames, which fails to detect the people in a conversation with more the threshold away and incorrectly counts the case where two pedestrians simple walk across each other.

Since the only code submitted with this report was the solution that used Faster RCNN detection with the Centroid tracking algorithm, the code for the other detection and tracking solution is found on the main GitHub repository located in the references below [16].

VII. CONTRIBUTION OF GROUP MEMBERS

Abhijit Hebsur: Implemented RCNN image detection algorithm, tested out the image detection using YOLO MobileNet. I also used centroid tracking for drawing trails.

Akhil Jain: Converted images to videos and used it to implement object detection with Contours and tracking with Kalman filters and Hungarian Algorithm. Detected people that are entering and exiting the frame in Task 1. Also, allowed the user to draw a bounding box in Task 2.

Denzil Saldanha: Implemented the Haar Cascade classifier. Worked on the Faster RCNN neural network implementation. Implemented the Centroid tracking algorithm. Added a uniform boundary box for groups of

pedestrians in Task 3. Worked on finding the results using the ground truth.

Peter Nguyen: Implemented the HOG detection feature descriptors with Linear SVM and the Non-Maximum Suppression (NMS) algorithm. Integrated the central tracking points in each pedestrian to calculate the number of pedestrians entering and exiting the boundary box in Task 2 and how many people are in a group or not in Task 3.

Ying Huang: Implemented YOLO v2 and v3 together with Deep Sort tracking method for Task 1. Refined the algorithm of the group detection and bounding box drawing for Task 3. Worked on implementing the Faster RCNN for Task 1, 2 and 3.

VIII. REFERENCES

[1] J. Cohen, "Computer Vision for tracking," Medium, 08-Apr-2020. [Online]. Available: https://towardsdatascience.com/computer-vision-for-tracking-8220759eee85.

[2] "Face Detection using Haar Cascades," OpenCV. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html.

[3] A. Kathuria, "What's new in YOLO v3?," Medium, 29-Apr-2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[4] Lee, N. Raj, M. F. Simalango, F. Khan, Tech Admin, and Mikael Fernandus Simalango Post, "What Object Categories / Labels Are In COCO Dataset?," Amikelive, 21-Apr-2020. [Online]. Available: https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/.

[5] Neuromation, "Tracking Cows with Mask R-CNN and SORT," Medium, 01-Jul-2018. [Online]. Available: https://medium.com/neuromation-blog/tracking-cows-with-mask-r-cnn-and-sort-fcd4ad68ec4f.

[6] A. Rosebrock, "Pedestrian Detection OpenCV," pyimagesearch, 09-Nov-2015. [Online]. Available: https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/.

[7] A. Rosebrock, "OpenCV center of contour," pyimagesearch, 01-Feb-2016. [Online]. Available: https://www.pyimagesearch.com/2016/02/01/opencv-center-of-contour/.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137–1149, Jan. 2017.

[9] L. Shapiro, "Object detection, deep learning, and R-CNNs," *University of Washington Computer Science & Engineering*. [Online]. Available: http://homes.cs.washington.edu/~shapiro/EE562/notes/Vision2-16.pptx.

[10] G. R. Valiati and D. Menotti, "A Preliminary Evaluation of Pedestrian Detection on Real-World Video Surveillance," *Int'l Conf. IP, Comp. Vision, and Pattern Recognition*, 2018.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.

[12] Yehengchen, "yehengchen/Object-Detection-and-Tracking," *GitHub*, 22-Dec-2019. [Online]. Available: https://github.com/yehengchen/Object-Detection-and-Tracking/tree/master/OneStage/yolo/deep_sort_yolov3.

[13] "YOLO vs Faster RCNN," *Everitt's blog*, 10-Aug-2018. [Online]. Available: https://everitt257.github.io/post/2018/08/10/object_detection.html.

[14] S.-H. Tsang, "Review: YOLOv2 & YOLO9000 - You Only Look Once (Object Detection)," Medium, 20-Mar-2019. [Online]. Available: https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65.

[15] J. C. Redmon, "Yolo V3 Weights," Joseph Chet Redmon. [Online]. Available: https://pjreddie.com/media/files/yolov3.weights.

[16] A. Hebsur, A. Jain, D. Saldanha, P. Nguyen, and Y. Huang, "COMP9517," GitHub, 24-Apr-2020. [Online]. Available: https://github.com/denzilsaldanha/comp9517.