COMP9517- Royal Flush Project Report

Akhil Kumar Yeruva
z5281223
z5281223@unsw.edu.au

Vishal Singh Niranjan
z5334390
z5334390@ad.unsw.edu.au

Sai Bhargav Kommuru
z5291858
z5291858@unsw.edu.au

Zeyu Hou
z5190728
z5190728@ad.unsw.edu.au

***Abstract*—In recent years, object detection and object tracking has become a significant research area in computer vision. This is because object tracking has a wide range of applications ranging from simple classification tasks to major surgical application procedures. So, designing a neural network model capable of performing detection and tracking with utmost accuracy and precision is critical. Our part of work in this paper includes detecting pedestrians in a video and keeping track of the recognized pedestrians throughout the video. So, object detection was done by using yolov4-tiny model as well as yolov4 model and object tracking by Deep sort using Kalman filters. Later in the project, some analytical tasks such as counting pedestrians, group formations, and pedestrians entering/leaving the video are also done. In this paper, we have developed two measurement metrics to find the efficiency of our used models for detection. Finally, we found that the yolov4 original model gave out a much better resultant score than another model. Finally, we sum up our project by pointing out insights and indicating future trends in this field.**

Index Terms – Pedestrian Tracking, Pedestrian Classification, Pedestrian Identification

I. Introduction

Computer Vision, from an engineering perspective, is an interdisciplinary field, which tends to automate and replicate the tasks of the human visual system through the help of images and videos. Computer Vision is used in a variety of influential industries such as the Utilities industry to classify different objects and tools, automotive industry for perceiving object's shape and size etc. These machine visualization tasks are achieved mainly with the use of two technologies: deep learning and convolutional neural networks. There are many categories of work in the field of computer vision and these tasks need to be performed in much less time with cameras and data given, and advanced algorithms and different models were considered to achieve the desired results. Further details about the methods and implementation will be explained in later sections.

The computer vision algorithms used in industries are widely based on pattern recognition. So, a massive amount of data is needed to train these models where the computer processes these images, learns about these objects and recognized patterns among them. Deep learning, which is a part of machine learning, provides an effective way of doing this, by making use of neural networks. The neural networks extract patterns from data samples and understand patterns by use of neurons and interconnections between them. Some of the widely used applications of computer vision are augmented reality, facial recognition, health sector (cancer detection etc.), also recently in agriculture field to monitor harvest of crops.

The focus of this paper is to develop and evaluate a method for detecting walking pedestrians in a video input data and be able to make useful analysis from it. This task requires the use of machine learning/deep learning methodologies to solve it. This problem can be categorized under object detection and object tracking, where the objects in this task are pedestrians. The main difficulty of this task is that the pedestrians are not stationary and can move in/out of the video input and the model developed should be able to recognize these pedestrians and not see them as separate objects. This task of recognition must be done, irrespective of size/shape of object, illumination on the object or of any deformations on the object. These restrictions can add some layer of difficulty to this task and that is why object tracking is still regarded as a highly active research area under computer vision.

The input dataset used is a public dataset and is from Segmenting and Tracking Every Pixel (STEP) benchmark which consists of 2 training sequences and 2 test sequences and is part of the ICCV21-Workshop Segmenting and Tracking Every Point and Pixel. These input videos extend the annotations used in the STEP benchmark and come with dense segmentation labels for every pixel in every frame. Every pixel belonging to the pedestrian class and is unique with other pedestrians came with its unique tracking id embedded in it. The 2 input train videos given also with their own mask frame images respective to their own video fame images. These mask frame images have pedestrians as red color and everything else as black. These mask images can be used to test the accuracy of our trained model.

The dataset used in this paper for training and testing can be found here: https://motchallenge.net/data/STEP-ICCV21/

II. Literature Review

(A) R-CNN

R-CNN was one of the first algorithms which was used for object detection. In CNN the system takes an input image and extracts around 2000 region proposals [1] and computes features for each of those proposals using a large CNN (Convolutional Neural Network) and it then classifies each region into a class specific linear SVM (Support Vector Machines) R-CNN achieves as average precision of 53.7 on a world-renowned challenge **PASCAL VOCC 2010** [2]**.** After the extraction of the region proposals the proposals must be labelled for them to train. Therefore, the author labels all the proposals having IOU (Intersection over union) of at least 0.5 with any of the ground-truth bounding boxes with their classes. However, all other region proposals that have an IOU of less than 0.3 are labelled as background and thus they are all ignored. [2]

Bounding box regression is a method popularly used to refine or predict localization boxes which is popularly used in R-CNN. The below deltas are to be predicted by CNN.

$$tx = (Gx - Px)/Pw$$
$$ty = (Gy - Py)/Ph$$
$$tw = log(Gw/Pw)$$
$$th = log(Gh/Ph)$$

So, (x,y) are center coordinates where w, h are width and height respectively. G and P stand for ground truth bounding box and region proposal respectively. The bounding box loss is only calculated for positive samples. The total loss is calculated as sum of classification and regression losses. [2]

Fundamentally, R-CNN can be classified as a 2-stage model. The initial model or stage is (RPN) who's job is purely finding the candidate in our case the "object" or "objects" in the scene along with an initial estimation of a bounding box of an object, then the second stages job is to utilize the local features around the proposed regions which are detected and determine the class or ignore them from the background and refine the bounding box. In simple words R-CNN finds the interesting regions and then for every interesting region it checks what objects are in the region and then it removes overlapping and low score detection and refines the bounding boxes.

Even though R-CNN is considered state of the art detection algorithm, R-CNN has its own disadvantages. Firstly, CNN is very slow cause of the way it was designed. In R-CNN first a Convnet is fine-tuned on region proposals using log loss. Then, it fits SVM to Convnet features by replacing Soft Max. [13] Only in the third stage regressors are trained. Training is heavy and is expensive in both space and time. R-CNN is slowly mainly because Convnet forward pass of each object proposals. There are about 2000 object proposals generated from each image. [3]

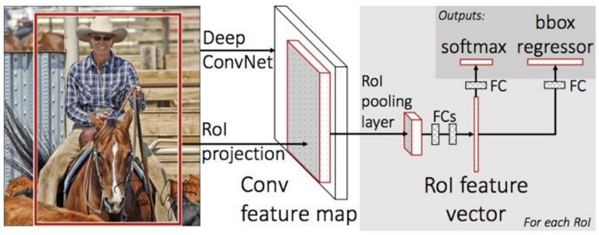

Fig-1. Architecture of R-CNN [4]

In recent years, there have been a lot of improvements in object detection algorithms and its performance. The best performing systems fare very complex ensembles combining multiple low level image features with high lever context from object detectors and scene classifiers. R-CNN achieves 30% relative performance improvement over CNN and other relatively older methods this was achieved by applying high-capacity convolutional neural networks to recognize region proposals to localize and segment the objects. The second is a pattern for training large CNN's when CNN's are labelled they are found to be highly effective during pre-training the network. R-CNN uses a combination of classic tools from computer vision and deep learning (CNN's and region proposals)

(B) Fast R-CNN

Fast R-CNN or Fast Region-based Convolutional Network method is an object detection method which is based on previously popular and highly efficient algorithm R-CNN. The reason "Fast R-CNN" is faster than R-CNN is because you don't have to feed 2000 region proposals to the CNN every time. Instead, the convolutional operation is done only once per image and a feature map is generated. [5]

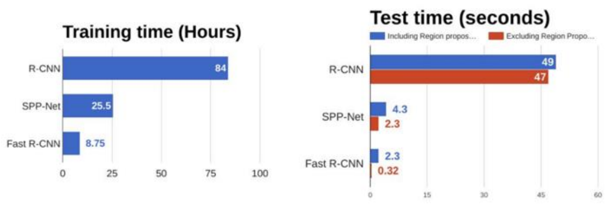

Fig-2. Comparison of object detection algorithms [5]

From the graphs below, you can deduce that Fast R-CNN is significantly faster than R-CNN. When you compare the compare the performance of the Fast R-CNN during testing, including region proposals slows down the algorithm drastically when compared to not using region proposals. Hence, Region proposals become a bottleneck in Fast R-CNN algorithm which affects its performance.

Fast R-CNN works faster using a method called ROI (Region of interest) projection. In ROI projection the region proposals in the original image are projected onto the output of the final convolution feature map. This map is then used by ROI pooling later. ROI pooling works by sub sampling ratio.

In simple terms consider we have a 18x18 image. After passing it through a CNN and max pooling it we get a 1x1 feature map. Then we will say we have a subsampling ration of 1/18. It is the ratio between scale of output feature map and the input image. Now that we understand sub sampling ratio the next step is to understand how it helps in ROI pooling. Let out input image be of size 688x920, and the feature map outputted from it be 43x58. We have a region proposal of size 320x128.

Sub Sampling Ratio = 58/920 = 1/16
New Bounding Box Coordinates = (320/16,128/16) = (20,8)
New Bounding Box Centre – (340/16,450/16) = (21,28)

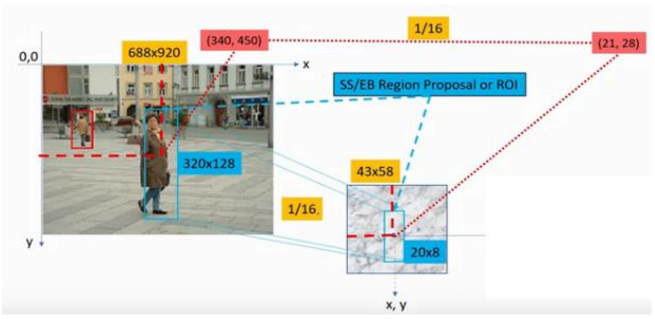

Fig-3. How subsampling helps ROI Projection [3]

After ROI Projection it goes through a process called ROI Pooling. In the process it takes a fixed sized feature map produced by the deep convolution network and an Nx5 Matrix list which represents a list of regions of interest, where N is the number of ROIs. From this it takes the corresponding regions from the input feature map and scales it to a pre-defined size. The scaling works by dividing the region proposals into equal-sized section then finding the largest value in each section and then copying these max values to the output buffer. This makes it very efficient than R-CNN. [3]

(C) Single Shot Detector (SSD)

Single shot detector is designed to detect object in real time. It has two variations: SSD300 and SSD512 with differences on the size of the input images. SSD has two components: a backbone model and an SSD head.

The backbone model is usually a pre-trained image classification network as a multi-scale feature extractor, which is built based on VGG16 to extract feature map. The feature extractor layers progressively decrease in size, and all the convolutional layer applied on each feature map is different.

The SSD head is convolutional layers applied to feature maps and the outputs are sixed set of detection predictions using a set of convolutional filters. For a feature size of $m \cdot n$ with p channels, the basic element for predicting parameters of a detection is a $3 \cdot 3 \cdot p$ small kernel that produces either a score for a category, or a shape offset relative to the default box position. At each of the $m \cdot n$ locations where the kernel is applied, it produces an output value. In summary, To detect objects, it then computes both the location and class scores using small convolution filters for each cell to make predictions.

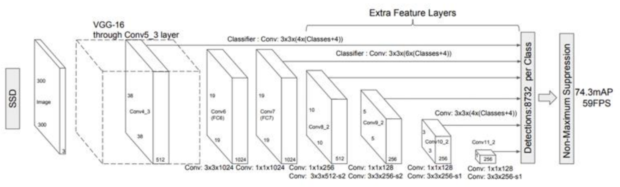
Extra Feature Layers
VGG-16 through Conv5_3 layer
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
SSD
Image
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS

Fig-4. Architecture of SSD Model [6]

(D) You Only Look Once (YOLO)

YOLO is an algorithm that detects and recognizes various objects in a picture. Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

The YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real time. As the name suggests, it only requires one single forward propagation through a neural network to detect objects. More detailed architectures will be presented in the methods section as we chose yolov4 as our model to detect objects.

YOLO has distinct advantages over other object detectors we mentioned above. Compared to other region proposal classification networks, it trains on full images and directly optimizes detection performance. Second, it reasons globally about the image when making prediction, which makes YOLO outperform Fast R-CNN by making less than half the number of backgrounds errors. Lastly, it learns generalizable representations of objects, which makes YOLO less likely to breakdown when applied to new domains or unexpected inputs.

III. Methods and Models

A. Pedestrian Detection and Bounding Box Calculation (tasks 1.1 and 1.3)

To perform the function of pedestrian detection, we considered several different models like R-CNN, Fast R-CNN, Efficient Det, YOLOv3, YOLOv4, YOLOv5 and SSD. But we finally picked YOLOv4 model for our system because of its superior performance as compared to other models.

YOLO stands for You Only Look Once and it was developed by Joseph Redmon. He described the YOLO model in detail in his paper You Only Look Once: Unified, Real-Time Object Detection [7]. Due to the huge success of the YOLO model, several versions (v3, v4, v5) of it were developed later.

We are using YOLOv4 model for pedestrian detection because of its better speed and accuracy as compared to other models. Also, for YOLOv4, we didn't need to convert input images into a different format as we do for YOLOv5. This is one of the main reasons we did not select YOLOv5 for our system.

YOLOv4 model is based on Darknet Neural Network and it consists of an input, backbone, neck, dense prediction and sparse prediction layers.

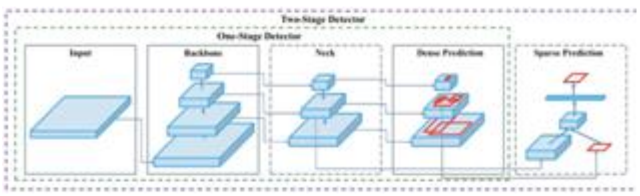

Fig-5. Architecture of YOLOv4 Model [8]

The process of YOLOv4 is divided into two parts: first it performs regression to identify objects and calculate their bounding boxes, and second it performs classification to classify the detected object into one of the class labels. Since in our system, there is only one type of object, pedestrian, we have only one class which will be used by the model.

YOLO model performs object detection by dividing an input image into several grids (or regions). Each cell of the grid has got the responsibility of identifying bounding boxes which can contain an object and the center of the bounding box lies within that cell. For each bounding box, the cell will predict the center of the box, width and height of the box, and the probability that the box contains an object (pedestrian).

It is possible that some bounding boxes are returned by the model which does not contain any pedestrian or contains only a part of a pedestrian. Such bounding boxes will have low probability of containing a pedestrian and we removed them in our model by applying a minimum probability threshold.

Also, it is possible that more than one bounding box contains the same pedestrian. In this scenario, one main bounding box will be covering the pedestrian entirely and there will be little blank space between the box boundary and the pedestrian. For the other overlapping boxes, the pedestrian would only come in some part of the bounding box. So, the main bounding box will have the highest probability as compared to other bounding boxes covering this pedestrian. So, we apply Non-Max Suppression (NMS) in this situation to check the overlapping boxes. NMS will remove any duplicate overlapping boxes and will keep only the highest probability bounding boxes [9].

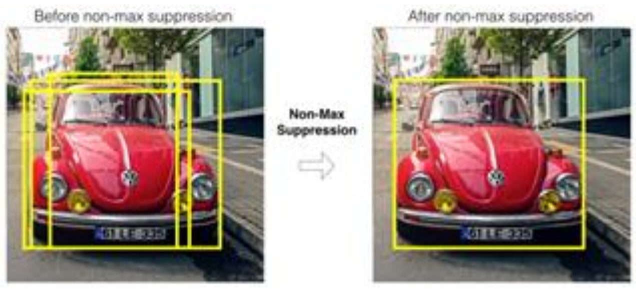

Fig-6. Non-Max Suppression [10]

This was the method we used to detect pedestrians and calculate bounding boxes for them. Each pedestrian will be identified by one bounding box only. Also, every bounding box would contain the centroid, width and height of the box which would be used in subsequent methods to track unique pedestrians and to group the pedestrians.

B. Pedestrian Tracking (Task 1.2)

To keep tracking the movement of pedestrians on all frames of the input video, the Deep SORT model is adopted which consists of three main components.

Track: A track is a class storing all the information for a single pedestrian tracking task, which contains Kalman state and associated velocities and features.

Tracker: The tracker is defined over some main attributes: metric, max_age, kf and tracks. Metric is the distance metric used for measurement to track association, max_age is the maximum number of missed misses before a track is deleted from the tracker, kf is the Kalman filter to filter the target trajectories in image space, and tracks is a list of active tracks at the current frame, which means the tracker is set up for multiple targets.

The tracker can predict each track stored in it, which propagates track state distributions one time step forward. The distance metric can be updated along with the given detections at the current time stamp.

Kalman Filter: The Kalman filter is made for tracking bounding boxes in this project. The 8-dimensional state space contains the bounding box center position, aspect ratio, height, and their respective velocities. The bounding box location is taken as direct observation of the state space (linear observation model).

In the main function, for each frame image, the tracker predicts the next tracking using Kalman filter and then updates the Kalman state taking the current centroid detections. Next it loops through the tracks to check if the bounding box is active. For each track with a unique track id, it draws a line with randomly assigned color connecting tracks on successive frames.

C. Pedestrian Count (Tasks 2.1, 2.2, 2.3, 2.4)

One of the biggest challenges of tracking and control systems is providing accurate and precise estimation of the hidden variables in the presence of uncertainty. Kalman Filter produces estimates of hidden variables based on inaccurate and uncertain measurements. Also, it provides a prediction of the future system state based on past estimations.

One of the analytical tasks done was to count the number of pedestrians in each frame. The total count of the pedestrians in each frame could be calculated with the help of detections given by the YoloV4 model. The detection algorithm segments different pedestrians in each frame and gives out an array of bounding boxes as explained before. Hence, the total count of pedestrians for a given frame is nothing but the length of the array of bounding boxes. This process is repeated for every frame and every time a new frame pops up, the count is updated accordingly based on detections given by YoloV4. The other task was to find the number of unique pedestrians throughout the input video. This task falls under object tracking, where every pedestrian must be tracked from frame to frame and be able to recognize him/her if they go out/come back to the current frame.

As discussed before, the deep sort tracker assigns unique id to each pedestrian and tracks them every frame from the moment they are registered. So, in every frame we get a list of tracking ids for every pedestrian detected and these tracking ids are passed as input to a hash set, which then gives out unique list of tracking ids. Hence, counting the number of these tracking ids in the hash set would give out the count of unique pedestrians present till the current frame. The deep sort tracker keeps updating itself with the detections by the detection algorithm every frame and hence, this counting of tracking ids is done every frame, to give count of unique pedestrians up till this current frame.

For more analysis, the user is given freedom to pause/un-pause the output video to see the previous two analysis of counts. The user can also draw a bounding box on the current paused frame and the video will show the user the number of pedestrians inside that drawn box. This is done by using the tracking bounding box given the deep sort tracker. For every frame, as we get the tracked bounding boxes of pedestrians, we simply see which boxes of pedestrians lie inside the bigger box drawn by the user. This way, we can get the count of pedestrians who are in the drawn user box for that particular frame. The user is also given the choice to pause the output video again and draw a new bounding box, and the algorithm will adapt itself to give the count of pedestrians lying inside this newly given box.

D. Group Formation and Destruction (Tasks 3.1 and 3.2)

To implement the task of identifying pedestrians if they are in group or if they are alone, we used the output from the tasks 1.1-1.3 where the pedestrians were identified and tracked using the Yolov4 algorithm. The bounding boxes were calculated in the previous tasks for the pedestrians.

As previously explained the pedestrians are identified and calculated every frame of the video. As part of the pedestrian tracking unique ID's are assigned to every pedestrian which makes it easier for the algorithm to keep track of unique pedestrians in the frame. The bounding boxes drawn from the previous implementation are used to track the pedestrians we extract the coordinates of the bounding boxes and the centroids and return a list of results.

With these results we begin the process of classification of these pedestrians. The method or approach we used is simple. To calculate if the pedestrians are in groups or singles, we used the output coordinates of the bounding boxes and gave weights or confidences to each of these bounding box coordinates and centroids from here we start calculating if the person is near another person or theoretically tell if the person is walking in a group or is walking alone. We do this by defining two pedestrians with their bounding boxes and calculate the distance between both the bounding boxes for each frame of the video. We extrapolate info from the bounding boxes i.e., Distance, Height, and Width. When the threshold reaches 0.6 or when they are close to each other with a distance threshold of 0.6 on the X- axis and 1.5 on the Y-axis they are classified as a group and when the bounding boxes threshold or distance is above this threshold then they are classified as alone of pedestrians who are not in a group. The distance between two bounding boxes is calculated every frame of the video

A formula was devised calculate if the pedestrian's bounding boxes are close enough to form a group this even works when there are multiple pedestrians. For example, when two people are in groups they are classified as groups and the color of the bounding boxes changes into blue to indicate that these two people are in group for that frame if they continue to stay close the boxes are still blue and on top of the bounding box a counter is shown this counter shows how many pedestrians are in the group. If a pedestrian leaves the group or crossed the threshold it is called a group destruction event and then the pedestrian is classified as single or alone, and the bounding box turns to yellow and a label of Alone is given.

If in the case over the next few frames instead of a group destruction a new pedestrian joins the frame this even is called group formation during which other than the two pedestrians an additional pedestrian is also considered in the group and the bounding box changes from blue to light blue and the count goes from 2 to 3. This works for pedestrian groups of up to 5 for now and as the count goes up in the group, the color and counter on the bounding box changes too.

E. Identification of Pedestrians Entering or Leaving (Task 3.3)

To carry out the task of identifying pedestrians entering or leaving the scene, we used the output of the pedestrian tracking task. As part of pedestrian tracking code, unique IDs were assigned to each pedestrian. This made the task of identifying entering pedestrians simpler.

While processing each input image frame, we kept track of all the tracking IDs and stored them in a dictionary. Whenever any tracking ID gets inserted into the dictionary for the first time, it indicates that that pedestrian has come into the video for the first time. At that time, we will assign a value, say 50, to that tracking ID in the dictionary and will highlight that pedestrian with a white arrow on top of the bounding box. If we observe that tracking ID again, we simply decrease the count by one and kept highlighting that pedestrian until the count becomes zero. We chose 50 as the number of frames for which we are highlighting any pedestrian entering the scene.

The task of identifying people leaving the scene was a little bit more difficult as we needed to check the upcoming frames to see when that person is going to leave the video. So, to perform this function, first we process all the input frames to find out the tracking IDs of all the pedestrians and stored them in a data structure along with the frame number in which we observed that tracking ID. Then we processed this data structure to check all the frame numbers and tracking IDs and checked backwards to find the last frame number in which a tracking ID appeared. After that we looped through all the images frames again and highlighted the last frames in which a tracking ID appeared.

We highlighted the leaving pedestrians by showing a red arrow on the top of their bounding box and they were highlighted for the last 50 frames in which they appeared in the video. Moreover, if any pedestrian appeared in the video for 50 frames or less, then they were just highlighted as entering pedestrians and not as leaving pedestrians.

IV. Discussion and Results

A. Pedestrian Detection and Bounding Box Calculation

Pedestrian detection functionality of our system works by first predicting all the bounding boxes that can contain an object and calculating the probability for each box that it covers a pedestrian. Then we filter out the bounding boxes that doesn't satisfy the minimum threshold criterion and keep only the highest probability boxes. We fine-tuned this minimum threshold parameter to try to detect different type of bounding boxes for the pedestrians. Finally, we picked the optimum value (0.2) that was giving us the best results for pedestrian detection and tracking.

The pedestrian detection functionality of our system works really well and it detects most of the pedestrians correctly. We used both the YOLOv4 and YOLOv4-tiny models to check and compare the results of our model.

YOLOv4 is the bigger model with a huge number of weights. The size of its weights file is more than 250MB. On the other hand, YOLOv4-tiny is the lightweight version of YOLOv4 and has fewer number of weights. The size of its weights file is around 24MB. Tiny model also runs a lot faster than the YOLOv4 model.

We tested the pedestrian detection functionality on both these models and found out that tiny model is able to detect more than 70% of the pedestrians while the YOLOv4 model is able to detect more than 85% of the pedestrians accurately. Detailed performance statistics are shown in 'Performance Evaluation' section under Conclusion.

However, both the models were not able to detect some pedestrians and they were getting missed during the detection phase. We investigated this issue further and found that it is happening when a person is not fully visible on the screen. It happens whenever any pedestrian goes behind any other pedestrian or behind any object.

The reason behind these pedestrians detection misses is that since these pedestrians are not fully visible on a frame, only some part of their body is captured by a bounding box and not the full body. So, when the YOLOv4 model calculates bounding box for them it assigns a low probability to them and they gets filtered out by minimum probability threshold and during the Non-Max Suppression phase. Possible solution for this issue is discussed in the 'Future Improvements' section under Conclusion.

Besides this small issue, pedestrian detection functionality is working well and detecting pedestrians with good efficiency.

B. Pedestrian Tracking and Unique Person Identification

Tracking is the problem of generating an inference about the motion of an object given a sequence of images. At the current stage, there are still difficulties in the tracking process, for example, noise in images, complex object motion/shape and partial and full object occlusions.

Object trajectory describes the course of a measured variable over time, for this situation, to obtain the object trajectory, it means identifying which detections in two successive frames of the video belong to the same pedestrian. Criteria for this can be based on distances between the boxes or features calculated from the pixel values within the boxes.

To implement this Pedestrian Tracking task, a pretrained deep SORT model is adopted to fit the data provided. Simple Online and Realtime Tracking (SORT) is a pragmatic approach to tracking multiple objects with a focus on simple effective algorithms. It performs Kalman Filtering in image space and frame-by-frame data association using the Hungarian method with an association metric that measures bounding box overlap. [11]

The pedestrian tracking functionality of our system is working well and able to track unique pedestrians effectively. But if a person does not show in the video anymore and appears after several frames then they are not identified as the previously tracked person and they are classified as a new person with a new tracking ID.

C. User Input Window and Pedestrian Count

The count for total pedestrians in the current frame and the count for unique pedestrians upto the current frame are calculated at every frame occurrence. These 2 counts are displayed in the top-middle of the output video and the user can see these counts updating themselves at every frame. The accuracy of these frames can be calculated with the help of Pedestrian Detection Efficiency(PDE) and Unique Pedestrian Detection Ratio (UPDR), which are discussed in much detail in the later sections. We have tested these counts using these metrics, by using two model networks: Yolov4-tiny and Yolov4. They were tested on 2 train videos given and their scores can be seen in the later sections. The best output score is given by Yolov4 with 86.09% PDE on *STEP-ICCV21-02 and 96.43*% PDE on *STEP-ICCV21-09* images.

After the user has input their own bounding box in the output video, then the algorithm successfully counts the pedestrians detected inside the user given box and outputs the count in the top-middle section of the output video. This algorithm has near perfect accuracy for counting pedestrians in the box and is also bounded by the detection models accuracy of finding all the pedestrians in the frame. So, this algorithm can be improved further by having a model with a very high accuracy rate.

D. Group Formation and Destruction

To recognize group formation and destruction we used the implementation of YOLO detection algorithm and deep sort to identify and track the pedestrians in the video. This implementation is then used to calculate their bounding boxes. We then extracted the bounding box coordinates and the centroids and calulcated the distance between then and that is how pedestrians were classified into groups or individuals. Our implementation successfully detects when a pedestrian is in group or when they are alone. We are successfully able to even get a count of pedestrians in the group.

Even though the implementation is very accurate and gives good results and can detect if the pedestrians are in groups and individuals there are limitations to it. As we can’t detect unique pedestrians, in the sense the algorithm can’t distinguish when a person goes out of frame and comes back in the frame and if they are the same pedestrian or a new pedestrian we can’t tell if the pedestrians are a group or just two random pedestrians being classified as groups. As in the video people keep going out of frame and keep coming back. We are limited with the dataset.

E. Identification of Pedestrians Entering or Leaving

The identification of pedestrians entering or leaving the scene functionality is working well for our system. It highlights any person entering or leaving by showing a white or red arrow on top of their bounding box, and these highlights quickly draws the attention of a viewer to these events. Moreover, these highlights are being shown for 50 frames which gives a viewer enough time to notice these variations. This number of 50 frames can also be updated in the code to increase or decrease the duration of frame highlighting.

However, there were a few instances when a same pedestrian is showed as more than one time entering or leaving the scene. This issue happens because whenever a pedestrian goes out of the video and comes back after a few seconds, then the pedestrian tracking software is not able to detect them as the same individual. That’s why such pedestrians are highlighted more than once in the whole video. Possible solution for this issue is discussed in ‘Future Improvement’ section under Conclusion.

V. CONCLUSION

In conclusion, our system effectively performs all the tasks for pedestrian detection and pedestrian tracking. It also completes the tasks of counting the number of pedestrians in a frame, number of unique pedestrians in the whole video, and the number of individuals in a user drawn box. The functionalities of grouping the pedestrians walking together and identifying the pedestrians entering or leaving the scene are also well defined. All these functionalities are designed in a way to fulfil all the project specifications completely.

Performance statistics of some of the tasks are mentioned in the following section. We were able to get only these metrics, and not any additional metrics, due to the lack of the information for training dataset labels. If we had data to identify the true positives, false positives, true negatives, and false negatives, then we would have used Accuracy Precision and other metrics for our system.

A. Performance Evaluation

To assess the performance of our system, we devised the below two metrics to gauge its results for pedestrian detection and pedestrian tracking problems.

1. Pedestrian Detection Efficiency (PDE)

$$PDE \ = 100 * \frac{Detected\ Total\ Pedestrian\ Count}{Actual\ Total\ Pedestrian\ Count}$$

PDE calculates the percentage of all the pedestrians in the video that were successfully detected by the system. It doesn’t check for the uniqueness of a pedestrian. So, in a perfect system, the value of PDE should be 100%. If any system is not able to detect all the pedestrians, then its PDE value would be less than 100%.

2. Unique Pedestrian Detection Ratio (UPDR)

$$UPDR \ = \frac{Detected\ Total\ Unique\ Pedestrian\ Count}{Actual\ Total\ Unique\ Pedestrian\ Count}$$

UPDR checks the uniqueness of each pedestrian, and it is used in pedestrian tracking part to determine how well a system can classify the same pedestrian in different frames as one person. It calculates the ratio of the detected total unique count to the actual total unique count in the whole video. So, in an ideal system, the UPDR value should be one. But if a system is not able to track all the pedestrians accurately, then it would consider the same pedestrian as two different persons in different frames, and the UPDR value would be greater than one.

We used the panoptic images and their labels provided in the training datasets STEP-ICCV21-02 and STEP-ICCV21-09 to find out the actual counts of the pedestrians in each frame and the total unique count. We also determined the detected counts for our own model and calculated the values of PDE and UPDR metrics. Moreover, we determined these metrics for both the YOLOv4-tiny and YOLOv4 models, and their values are shown in the table below.

Model	**Performance of Models Tested**		
	Dataset Tested	***PDE***	***UPDR***
YOLOv4-tiny	STEP-ICCV21-02	70.64%	1.48
YOLOv4-tiny	STEP-ICCV21-09	72.25%	1.57
YOLOv4	STEP-ICCV21-02	86.09%	1.88
YOLOv4	STEP-ICCV21-09	96.43%	3.20

Table 5.1 Performance of Models

*Please note that in our submission, we are only submitting the weights and configuration files for YOLOv4-tiny model because the size of the YOLOv4 weights file was greater than 250MB and we were not able to upload it.

As seen in the above table, our system performed very well and were able to detect more than 70% of pedestrians for the YOLOv4-tiny model and more than 85% of pedestrians for the YOLOv4 model.

B. Future Improvements

Even though the performance of our system were up to the standards mentioned in the specifications, there were some limitations in our system, and it was not able to achieve very high scores (>90%) on the performance tests. These shortcomings and their possible solutions are discussed below.

- Our system was not able to detect some pedestrians if major part of their bodies were hidden behind other people or obstacles. This issue can be fixed by training the system on a dataset which have labels for partial pedestrians or pedestrians with some of their body parts hidden. Then the system would get the ability to identify and detect such pedestrians.

- Our system was able to track most of the pedestrians correctly, but there were some instances when if a pedestrian disappears from the video and comes back again after several frames, then the system would recognize that pedestrian as a new individual with a new tracking ID. This issue is more complex than the previous one, but it could be tackled by also identifying pedestrians based on their appearances and body features in the whole video.

- As per the provided specifications, we designed our system to classify pedestrians as individuals or groups if they stay together for more than one frame only. But since we are checking the distances between individuals for only 1-2 frames, it happened that even when some other individuals crossed the grouped individuals, they were considered as part of the group. This problem could be fixed by considering the closeness requirements of groups for several frames (at least 60-80) and then classify the individuals as grouped if they stay together for a major part of the checked frames.

- As per the provided specifications, we designed our system to detect and track only pedestrians. But it is also possible to modify the system to detect other objects besides pedestrians. To achieve this purpose, we would need a dataset which have images and labels classifying those new objects.

Overall, our system fared very well and was able to perform all the tasks (pedestrian detection, tracking, counting, grouping, etc.) with high efficacy. The minor issues were mainly because we didn't have a huge training dataset of pedestrians which can identify partial pedestrian images. By training the system on a dataset of tens of thousands of images and by employing a neural network with even greater number of weights, the system should be able to achieve efficiencies greater than 90% or even 95%.

VI. Contribution of Group Members

- Akhil Kumar Yeruva (Z5281223):

  Implementing code for Group formation and destruction events. Calculating number of pedestrians in groups and implementing different color bounding boxes as per the count of pedestrians in the Group. Corresponding parts in paper, Literature Review, Editing Paper. Fixing References. Formatting Paper.

- Vishal Singh Niranjan (Z5334390):

  Researched several models initially to find out which model works better for our system and developed the YOLOv4 model which is used to detect all the pedestrians and count the number of pedestrians. The output of this model is used by all the subsequent tasks to achieve their goals. Completed the code and video for my tasks 1.1, 1.3 and 3.3. Developed the performance metrics for evaluating pedestrian detection and pedestrian tracking functionalities. Integrated the groups code with the main merge section on github and added comments and references. Added sections for tasks 1.1, 1.3 and 3.3 for the sections 'Methods and models' and 'Discussion and Evaluation' and 'Conclusion' section. Also, edited and formatted the final report for this project.

- Sai Bhargav Kommuru (Z5291858):

  Helped in implementing the object tracking deep sort algorithm. Implemented the code for counting pedestrians in each frame, counting unique pedestrians and also counting pedestrians in user given box. Contributed for Abstract, Introduction section, complete task 2 in "Methods section" and completed task 2 in "Discussion and results section". Set up git repository and corresponding file structures of code for easier collaboration while implementing our own section.

- Zeyu Hou (Z5190728):

  Implemented Pedestrian tracking tasks and corresponding parts, Edited video and collated them into one and submitted. Contributed for SSD and Yolo in the report. Edited paper

VII. References

[1] P. Arbelaez, J. Pont-Tuset, J. Barron, F. Marques, and J. Ma- ´ lik. Multiscale combinatorial grouping. In CVPR, 2014

[2] C. Chingis, "Object Detection Explained: R-CNN", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/object-detection-explained-r-cnn-a6c813937a76.

[3] A. Mohan, "Review on Fast RCNN", *Medium*, 2022. [Online]. Available: https://medium.datadriveninvestor.com/review-on-fast-rcnn-202c9eadd23b.

[4] R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.

[5] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms", Medium, 2022. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[6] W. Liu et al., "SSD: Single Shot MultiBox Detector", Computer Vision – ECCV 2016, pp. 21-37, 2016. Available: 10.1007/978-3-319-46448-0_2

[7] Joseph Redmon. You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640

[8] Roboflow: YOLOv4 Darknet. https://models.roboflow.com/object-detection/yolov4

[9] Adrian Rosebrock: YOLO object detection with OpenCV. https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

[10] Lentin Joseph: A Gentle Introduction to YOLO v4 for Object detection. https://robocademy.com/2020/05/01/a-gentle-introduction-to-yolo-v4-for-object-detection-in-ubuntu-20-04/

[11] N. Wojke, A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric", 2017 IEEE International Conference on Image Processing (ICIP), 2017. Available: 10.1109/icip.2017.8296962