Detection of pedestrians and analysing the movements in videos

Group Name: PwxLxJzhLpbXx

WENXUN PENG
University of New South Wales
z5195349

LI XU
University of New South Wales
z5266680

ZIHAO JIANG
University of New South Wales
z5241178

PEIBO LI
University of New South Wales
z5219991

XING XING
University of New South Wales
z5142063

I. Introduction

Analysing pedestrians (or humans) in videos is one of the most common applications of Computer Vision. A wide range of practical scenarios ranging from simple domestic surveillance to self-driving cars thrive on the success of pedestrian detection, tracking and advanced analysis of motion.

In this project, we are required to perform pedestrian detection,tracking and behavior recognition on each frame of pictures, which involves the problems of object detection, object tracking and object recognition.

In computer vision, the detection problem can be viewed as a special case of image classification: given a candidate image window, the detection seeks to classify the latter as a pedestrian or non-pedestrian. Different techniques are used for detection, which can be classified into 6 main categories[1]: visual appearance-based detection, motion-based detection, spation-temporal features detection, 3D feature detection models, deep learning methods and attention windows detection.

While detection refers to finding the presence or absence of pedestrians at locations and scales in images, recognition here refers to the recognition of attributes of pedestrians given such detections. Recognition takes as input the localised window of visual or other sensor data forming the detection, and yields as outputs some information about the particular pedestrian detection.

Pedestrian tracking is the process of updating belief about a pedestrians location from a temporal sequence of data[2]. A pedestrian track is a sequence of a pedestrians locations over time. A pedestrian pose track is a sequence of a pedestrians body pose states over time. When multiple pedestrians are present, tracking requires knowing which pedestrian is which over time, associating the identities of the pedestrians with tracks. This is challenging problem for humans if their tracks overlap or disappear behind obstacles, and appears to require high level social intelligence and knowledge to guess what most likely happened when tracks are temporarily hidden.

For this component we are provided a sequence of images (extracted from a video) from The Eleventh IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2009) [3]. This data sets have 795 images without labels, and each image represent a single frame of the whole images sequence. Since there is no label image, it would be difficult to train a new neural network to perform detection and tracking tasks. Therefore, in this project, consider using traditional feature extraction methods or using pre-trained neural networks for detection and tracking.

II. Literature Review

In this project, we used deep learning methods to solve the pedestrian detection problem. The 2D and 3D detection models presented so far rely on a two stage process of feature extraction followed by classification. In deep learning these are merged into a single step classifying the raw data directly using neural networks, which are organized in hierarchical models learned from data. An advantage of the deep learning approach is that features can be learned, instead of manually “crafted”, which reduce errors on specific tasks rather than only on detection. The high performance of deep learning models comes at a price, they require larger training data (sometimes millions examples), longer training times (up to several days) and their computational cost is more important than for old-fashioned detectors e.g HOG-SVM.

The main application of deep learning is the convolutional neural network, which is a special neural network structure. Convolutional neural network mainly includes convolutional layer, pooling layer, dropout layer and fully connected layer. Because the low-level and high-level features of the image can be well extracted, convolutional neural networks have been proven to be used in many fields. R-CNN [4] is a special neural network that has been modified on the basis of the convolutional neural network. It combines region proposals with CNNs in order to solve detection problem by operating within the “recognition using regions” paradigm. The system first takes an input image, extracts around 2000 bottom-up region proposals.Then, it computes features for each proposal using a large CNN. Finally, it classifies each region using class-specific linear SVMs.

Based on R-CNN, a network structure called YOLO was proposed. YOLO is a deep learning based software for object detection and classification. Unlike R-CNN, YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. It is able to detect about twenty different classes including people, cats and dogs. In this project, we use YOLO v3 to perform pedestrian detection.

A. YOLO

YOLO[5] is a neural network used for detection and classification after R-CNN was proposed. YOLO is using a

single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

This detection network divides the input image into an S ×S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities. The final prediction can be an $S \times S \times (B * 5 + C)$ tensor. By using this tensor, detection can be easily performed.

The network has 24 convolutional layers followed by 2 fully connected layers. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

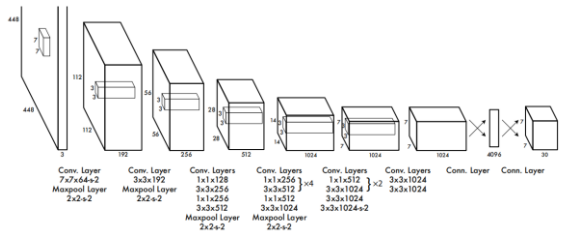


Fig. 1. YOLO Network Structure.

B. YOLO v2

In order to improve accuracy of detection, YOLO v2 was proposed based on YOLO. Compared with YOLO, YOLO v2 has higher detection accuracy and faster detection speed.

YOLO v2 performs batch normalization on all convolutional layers. This improvement not only speeds up the convergence of the model, but also increases the mAP by 2%, while reducing overfitting.

YOLO v2 uses anchor boxes method to perform prediction. The author proposes to use K-means as a clustering method to select anchors. The purpose of using K-means at the same time is to choose a good anchor so that IoU has a higher value.

YOLO v2 also use methods like high resolution classifier, fine-grained features and multi-scale training to improve its performance on detection.

YOLO v2 introduces a new classification network named Darknet-19, including 19 convolutional layers and 5 maxpooling layers. The detailed network structure is shown in the following figure:

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

Fig. 2. Darknet-19 Classification Network.

C. YOLO v3

YOLO v3[6] doesn't have much innovation, it is mainly to use some good solutions to integrate into YOLO. However, the effect is still good. On the premise of maintaining the speed advantage, the prediction accuracy is improved, especially the ability to recognize small objects.

The main improvements of YOLO3 are: adjusting the network structure; using multi-scale features for object detection; using logistic for object classification rather than softmax. The detailed network structure is shown in the following figure:

	Type	Filters	Size		Output	
	Convolutional	32	3	3	256	256
	Convolutional	64	3	3 / 2	128	128
1	Convolutional	32	1	1		
	Convolutional	64	3	3		
	Residual				128	128
	Convolutional	128	3	3 / 2	64	64
2	Convolutional	64	1	1		
	Convolutional	128	3	3		
	Residual				64	64
	Convolutional	256	3	3 / 2	32	32
8	Convolutional	128	1	1		
	Convolutional	256	3	3		
	Residual				32	32
	Convolutional	512	3	3 / 2	16	16
8	Convolutional	256	1	1		
	Convolutional	512	3	3		
	Residual				16	16
	Convolutional	1024	3	3 / 2	8	8
4	Convolutional	512	1	1		
	Convolutional	1024	3	3		
	Residual				8	8
	Avgpool		Global			
	Connected		1000			
	Softmax					

Fig. 3. YOLO v3 Network Structure.

III. METHOD

A. Task1a detection:

YOLO detection:

1. YOLO theory:

According to Redmon et al. (2015), YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. As its name, using YOLO system, you only look once (YOLO) at an image to predict what objects are present and where they are. YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes.

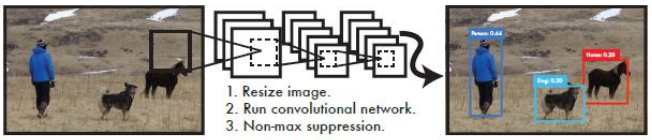


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection:

First, YOLO is extremely fast. Since YOLO frames detection as a regression problem it doesn't need a complex pipeline. It just simply run neural network on a new image at test time to predict detections. Since we are detecting real-time video and the speed of detection is very important, YOLO is suitable for real-time video.

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. In addition, YOLO achieves more than twice the mean average precision of other real-time systems. Even compared to Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

Although YOLO still lags behind the latest detection system in accuracy, it can quickly identify the objects in the image, which is enough for our real-time video detection system.

The above introduces the advantages of YOLO, we used a more complete version of YOLOv3 model as mentioned before.

2. YOLO implementation:

YOLO is a deep learning algorithm, so itself doesn't need any installation, what we need instead is a deep learning framework where to run the algorithm.

Here I'm going to describe the 3 most used and known frameworks compatible with YOLO and the advantages and disadvantages of each one:

Darknet: it's the framework built from the developer of YOLO and made specifically for yolo. **Advantage:** it's fast, it can work with GPU or CPU. **Disadvantage:** it only works with Linux operating system.

Darkflow: it's the adaptation of darknet to Tensorflow (another deep learning framework). **Advantage:** it's fast, it can work with GPU or CPU, and it's also compatible with Linux, Windows and Mac. **Disadvantage:** the installation it's really complex, especially on windows

Opencv: opencv has a deep learning framework that works with YOLO. Just make sure opencv 3.4.2 at least. **Advantage:** it works without needing to install anything except opencv. **Disadvantage:** it only works with CPU, so you can't get really high speed to process videos in real time.

According to the above, we want to make the deployment of the environment as simple as possible and we can tolerate lower process speed, so we choose to use opencv.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Next we will introduce how to deploy YOLOv3 in opencv.

First, loading the algorithm. We need three files to run YOLOv3 algorithm:

Weight file: it's the trained model, the core of the algorithm to detect the objects.

Cfg file: it's the configuration file, where there are all the settings of the algorithm.

Name files: contains the name of the objects that the algorithm can detect.

Second, using "cv2.dnn.blobFromImage()" function to get blob. Since we can't use right away the full image on the network, we need it to convert it to blob. It's used to extract feature from the image and to resize them. YOLO accepts three sizes:

320×320 it's small so less accuracy but better speed

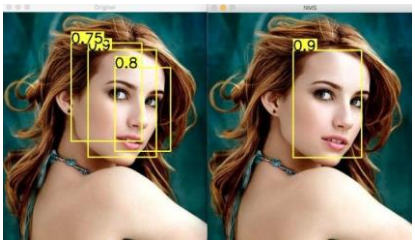
609×609 it's bigger so high accuracy and slow speed

416×416 it's in the middle and you get a bit of both. (so we used this sizes in our project)

Third, Input the blob into the already loaded YOLO model and get the test result.

Fourth, we set a threshold confidence of 0.5, if it's greater we consider the object correctly detected, otherwise we skip it. The threshold goes from 0 to 1. The closer to 1 the greater is the accuracy of the detection, while the closer to 0 the less is the accuracy but also it is greater the number of the objects detected.

Fifth, as mentioned before, when we perform the detection, it happens that we have more boxes for the same object (different confidences with the same object), so we should use another function to remove this "noise". Thus, we should use Non-Maximum Suppression (NMS). For example, if we don't use the NMS algorithm, we will get the result showed below:



The left image doesn't use the NMS algorithm, and we can get three boxes in the images (the confidence is 0.75, 0.9, 0.8 respectively). And the right image uses the NMS algorithm, so it just gets the maximum confidence.

NMS Algorithm: Input is a list of Proposal boxes B, corresponding confidence scores S and overlap threshold N. And output is a list of filtered proposals D. First, select the proposal with highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty). And then compare this proposal with all the proposals — calculate the IOU (Intersection over Union, IOU calculation is actually used to measure the overlap between two proposals.) of this proposal with every other proposal. If the IOU is greater than the threshold N, remove that proposal from B. Next, taking the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D. And then once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes which have high IOU than threshold. In the end, this above process is repeated until there are no more proposals left in B.

Finally, we just set the label to "person" in YOLO model (YOLO model can detect other object such as car, bicycle, etc.) and draw the bounding boxes in the images.

B. Task1b tracking:

Tracking is a little bit different with detection. We use tracking algorithm to predict the pedestrians path. It can establish the identity of the objects across frames. And object detection may fail but it may still be possible to track the object because tracking takes into account the location and appearance of the object in the previous frame.

There are two methods in task1b:

1 OpenCV CSRT tracking: There are eight separate object tracking algorithms built right into OpenCV, BOOSTING Tracker, MIL Tracker, KCF Tracker, CSRT Tracker, MedianFlow Tracker, TLD Tracker, MOSSE Tracker, GOTURN Tracker

In these eight trackers, the CSRT, KCF and MOSSE tracker have outstanding advantages. And for these three algorithms:

CSRT: when higher object tracking accuracy is needed and it may be slower FPS throughput.

KCF: when faster FPS throughput is needed but it may handle slightly lower object tracking accuracy.

MOSSE: when pure speed is needed.

Thus, considering the tradeoff of process speed and accuracy, we choose the CSRT algorithm. Let's briefly introduce the CSRT algorithm. CSRT algorithm uses the discriminative correlation filter with channel and spatial reliability (DCF-CSR), and uses the spatial reliability map for adjusting the filter support to the part of the selected region from the frame for tracking. This ensures enlarging and localization of the selected region and improved tracking of the non-rectangular regions or objects. It uses only 2 standard features (HoG and Color names). It also operates at a comparatively lower fps (25 fps) but gives higher accuracy for object tracking.

Next, we will briefly introduce the procedures:

First, using YOLO detection to locate objects in the first frames. OpenCV provides a function called selectROI that pops up a GUI to select bounding boxes (also called a Region of Interest (ROI)). And in the Python version, selectROI returns just one bounding box. Thus, we need a loop to obtain multiple bounding boxes.

Second, initializing the MultiTracker. We first create a MultiTracker object and add as many single object trackers to it as we have bounding boxes. As mentioned before, we use the CSRT single object tracker. The CSRT tracker is not the fastest but it produces the best results in many cases we tried.

Finally, update MultiTracker & Display Results. Our MultiTracker is ready and we can track multiple objects in a new frame. We use the update method of the MultiTracker class to locate the objects in a new frame. Each bounding box for each tracked object is drawn using a different color.

2 OpenCV centroid tracking: This is the main algorithm we used in our project since it is easy to understand and CSRT just provides a port to implement which is hard to optimize. In addition, we find that centroid tracking can achieve good performance.

Step 1: Accept bounding box coordinates and compute centroids. The centroid tracking algorithm assumes that we are passing in a set of bounding boxes (x, y)-coordinates for each detected object in every single frame. These bounding boxes can be produced by any type of object detector we would like (Haar cascades, HOG + Linear SVM, SSDs, Faster R-CNNs, YOLO (here we used) etc.), provided that they are computed for every frame in the video. Once we have the bounding box coordinates we must compute the "centroid", or more simply, the center (x, y)-coordinates of the bounding box. Since these are the first initial set of bounding boxes presented to our algorithm we will assign them unique IDs.

Step 2: Compute Euclidean distance between new bounding boxes and existing objects. For every subsequent frame in video stream we apply Step 1 of computing object centroids; however, instead of assigning a new unique ID to each detected object (which would defeat the purpose of object tracking), we first need to determine if we can associate the new object centroids with the old object centroids. To accomplish this process, we compute the Euclidean distance between each pair of existing object (old object) centroids and input object (new object) centroids.

Step 3: Update (x, y)-coordinates of existing objects. The primary assumption of the centroid tracking algorithm is that a given object will potentially move in between subsequent frames, but the distance between the centroids for frames F(t) and F(t+1) will be smaller than all other distances between objects. Therefore, if we choose to associate centroids with minimum distances between subsequent frames we can build our object tracker. If there is a lonely point that its Euclidean distance is not associate any other point, we will register new objects which is step 4.

Step 4: Register new objects. In the event that there are more input detections than existing objects being tracked, we need to register the new object. "Registering" simply means that we are adding the new object to our list of tracked objects by: 1. Assigning it a new object ID. 2. Storing the centroid of the bounding box coordinates for that object. And then we can then go back to Step 2 and repeat the pipeline of steps for every frame in our video stream.

Step 5: Deregister old objects. In the last step, we deregister old objects when they cannot be matched to any existing objects for a total of N subsequent frames.

C. *Task1c real-time counting:*

Based on task1b, using tracking algorithm to track all pedestrians showed on the images. Since the tracking algorithm mentioned before has a shortcoming that is difficult to overcome which is handling the occlusion. Specifically, some pedestrians will be completely or partially occluded by other objects (such as lamp posts, other pedestrians, etc.), which will cause neither the tracking algorithm nor the detection algorithm to recognize them. Thus, if we directly calculate the number of bounding box as their real-time counting for each frame, it will be incorrect. (some people are occluded, so their bounding box cannot be detected).

Therefore, we set two counter to count the real-time pedestrians from the each frame. The idea is to do two counts: one is to directly count the bounding box, and the other is to add or subtract a value per frame. The initial value of the second count is 3 (since there are 3 pedestrians in the first frame). And then detecting whether there is a new pedestrian walking into the frames. This algorithm is just like task 1b registering new object, and then getting the centroid of pedestrians. If x <50 or x> 726 or y <100 or y> 500 (the resolution of the frame we used is 768 * 576, so the points that satisfy the above range of x, y should all be at the edge of the frame), the second count value will plus 1 which means that there is a pedestrian walking into the frame (if there are two or more pedestrians walking into the frame and satisfying the centroid range, the second count will plus two or more values). Otherwise, the second count is counted as the recovery of the blocked object (Since if there is a occlusion, the first bounding box count will be reduced by 1). Similarly, when an object is deregistered at boundary of the frame, the second counts will be reduced by 1. Finally, combining two counts. The first one is used when there is no object deregister in a frame, and the second count is set to the same value of the first. When there is pedestrian deregister in this frame, using the second count. And we will not deregister the pedestrians unless they disappear in the next 5 frames (we test many other values and find 5 is the best value). Therefore, a pedestrian that is only blocked by one or more frames (less than 5) will not affect the count. However, it cannot solve the problem of being blocked for many sequence frames.

D. *Task 2a / 2b entering and moving out of the bounding box:*

The idea we use in this task is based on task1b, so we will briefly introduce the idea. First, getting the drawing bounding box from users. Second, detecting and tracking all pedestrians showed in the each frame. Finally, if the previous frame of this pedestrian is not in the drawing box, this pedestrian in this frame is regarded as entering. Otherwise, it is considered leaving. (If the drawing bounding box includes some pedestrians at the beginning, these pedestrians will not be counted into entering. However, if they leave the drawing bounding box, they will be counted into leaving.)

E. *Task 3a / 3b group:*

In this task, it is like task 1b. First, detecting and tracking all the pedestrians of each frame.

Second, if two bounding boxes of the corresponding pedestrians have an intersection or are very close (The centroids corresponding to two pedestrians differ by 50 pixels since we test many values and get 50 pixels), calculating the vector of their trajectory separately. For pedestrians who have intersections or are close, if they appear together for more than 20 frames, calculating the direction of its last 20^{th} frame to the last frame. Specifically, calculating the motion trajectories of the two pedestrians respectively. And the motion trajectories can be calculated by the corresponding pedestrians' centroid of the last 20^{th} frame and the last frame and link them. For example, using the centroid of the last 20^{th} frame as the starting point S, and the centroid of the last frame as the end point E, the two points are combined (S, E), which is the vector of the pedestrian motion trajectory. Similarity, if it is less than 3 frames, the last two frames will be used to calculate the motion trajectory. As for other situation, calculating the direction from the first frame to the last frame. For pedestrians who appear together in many frames, we found that taking the last 20th frame to the last frame can get relatively good results and it is reasonable (care about the latest situation). Thus, we calculating the direction of last 20^{th} frame to the last frame.

Third, if the angle between their vectors (motion trajectory) is greater than 45 degrees, it will not be considered as a group. If it is less than 45 degrees, there are two situations: 1. If the distance between the start and end points of the two vectors is greater than 15 pixels (in order not to calculate the pedestrians who just walk around in place instead of walking together) 2. If the distance between the start and the end point of the two vectors is less than 1 (pedestrians who stay in place). These two situations are counted as groups.

Finally, using the two bounding boxes from the group to form a new bounding box. The new bounding box is formed by taking the largest length and the largest width of the two bounding boxes. And then, the algorithm is a bit like task 1b. Calculating the centroid of each group and matching the new group with the existing group through the distance of their centroids. And the new group will be registered if there is not a match. In addition, if the existing group has no match, its disappear count will plus 1 (in fact, it is the same as deregister in task 1b). In task 1b, it is tracking the pedestrians but here, it just like tracking the group to detect the group.

IV. Experimental Setup

A. *Experimental Setup*

Dependencies:

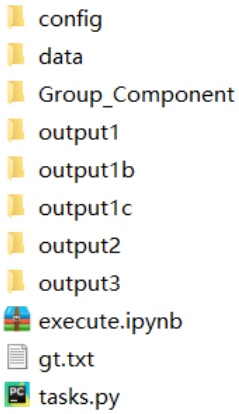
Python3, Opencv ≥ 3.4.2, Numpy, Matplotlib, os, math

Command format:

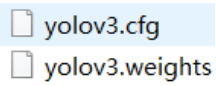
Using jupyter notebook open the execute.ipynb and all of the tasks in there.

Param set: Since we introduce and explain the value selection of all the key parameters in the Method part, we skip it here.

File directory:



The above is the file structure. The "config" folder contains the following two files:



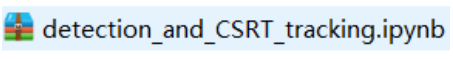
And the "data" folder contains "coco.names" file showed below.

These three files are YOLOv3 model file as mentioned in Method task1a. "yolov3.weights" is the trained model, the core of the algorithm to detect the objects. And as mentioned before, we used YOLOv3, if you want to use other version of YOLO, you should find other ".weights" files to get the trained model. "yolov3.cfg" is the configuration file, where there are all the settings of the algorithm. A ".cfg" file corresponds to a ".weights" file. If the corresponding file is not downloaded, for example, "yolov3.weights" and "yolov3-tiny.cfg" are used, it doesn't work. "coco.names" contains the name of the objects that the algorithm can detect. We just use "person" label here. More information on the YOLO model and configuration files can be found on the YOLO website:

https://pjreddie.com/darknet/yolo/

"gt.txt" is a ground truth files we find and it will be introduced in the next part. "tasks.py" contains all classes we used. And "execute.ipynb" is the main file that can get all tasks result from it. "Group_Component" contains all of original frames we used. "output1", "output1b", "output1c" represent respectively the result of task 1a, 1b and 1c. And the "output2" and "output3" represent the result of task2 and task3. All these folders have 795 frames with corresponding results showed (like the number of real-time count, trajectory tracking and bounding boxes for detection.)

And we can put the OpenCV CSRT implementation method in the "detection_and_CSRT_tracking.ipynb".



All the model files we need can be found in this google drive links:

https://drive.google.com/drive/folders/16aExpSDUwe_ATcdw0e6PTJp9uyX9NIui?usp=sharing

B. Evaluation Methods

Task 1 & 2:

We find a ground truth about our dataset from this website:

https://motchallenge.net/data/2D_MOT_2015/

As mentioned before, "gt.txt" file is the ground truth file. The format of the file:

frame, id, bb_x, bb_y, bb_width, bb_height, conf, x, y, z

frame: the number of frames, in our project, the range is 1 to 795 since we have total 795 frames in the video sequence.

id: pedestrians of uniquely identify.

bb_x, bb_y, bb_width, bb_height: they respectively represent the x coordinate, y coordinate, width and height of the bounding box where the pedestrian is located.

conf: the confidence that pedestrians fall in this bounding box. Since it is ground truth file, all of confidence are 1.

x, y, z: These values are ignored since they are used in 3D and we just need 2D.

For example, the first three lines from "gt.txt" showed below:

1,9,499,158,31.03,75.17,1,-4.1554,-7.3591,0

1,15,258,219,32.913,88.702,1,-11.306,-5.5995,0

1,19,633,242,42.338,81.074,1,-9.0323,-12.587,0

The first param "1" represents the first frame. The second param "9", "15" and "19" represent the ids of the three pedestrians on the first frame. The next params "499", "158", "31.03", "75.17" from the first line represent the bounding box of the person whose id is 9. We only used the frame, id, bb_x, bb_y, bb_width, bb_height in ground truth, so other params we will not explain anymore. And we can use bb_x, bb_y, bb_width, bb_height to calculate the centroid of the bounding box (centroid is the center coordinate of the bounding box: (bb_x + bb_width / 2, bb_y + bb_height / 2) as mentioned from the Method task 1b).

Since in our project, the performance of all algorithms implemented by task1 and 2 is related to the tracking implemented by task1b (task1a is a little bit different, task1a is only using the model and algorithm of YOLOv3. And the performance of YOLOv3 has been proven by many other people, but we used the same algorithm to evaluate the task1a since detection can also get the centroid of bounding boxes). Specifically, because both task1c real-time counting (if the tracking algorithm performance is better, we do not need to set two counts to deal with the occlusion situation, we can directly calculate the bounding box.) and task2 drawing bounding box counting entering and leaving (task1b is full-tracking and task2 is local tracking) are based on task1b's tracking algorithm, we only need to know the performance of task1b tracking algorithm that can know the performance of other algorithms. Thus, we focus on performance of task1b.

From the above, we can know that a pedestrian can get a centroid in each frame when tracking algorithm is used, and then a centroid can be calculated from the ground truth. Thus, we use the centroid of the bounding box we tracked to compare the bounding box calculated by the ground truth centroid. If the difference between the two x of the two centroid and the difference between the two y coordinates are both within 10 pixels, they are considered the same bounding box. That means we track the same person with the ground truth. And then, we used two methods to evaluate the performance of the tracking algorithm:

Accuracy: Accuracy is defined as: the ratio of the number of samples that the classifier correctly classifies to the total number of samples for a given test data set. More specifically, in this project, the total number of samples is the number of all bounding boxes from all frame in the ground truth. The correctly classified pedestrians are the same pedestrians we tracked and ground truth detected as mentioned before. It is like total number, we just count the bounding boxes of correctly classified pedestrians from all frames.

F1 score: However, Accuracy is not always effective, for example, Google crawled 100 pages of argcv, and it has a total of 10,000,000 pages in the index. Randomly selected a page and classified, is this a page of argcv? If we judge our work by accuracy, then we will judge all pages as "pages that are not argcv", because we are very efficient (return false, one sentence), and accuracy has reached 99.999% (9,999,900 / 10,000,000). Therefore, we need another method to evaluate the performance which is called F1 score.

Before introducing F1 score, we need to introduce the precision and recall. The definition of **recall** is the number of true positives divided by the number of true positives plus the number of false negatives. And the definition of **precision** is the number of true positives divided by the number of true positives plus the number of false positives. However, **precision** and **recall** affect each other. In general, a good performance should have both high precision and high recall. However, in fact, if the precision is high, the recall will be low, and vice versa. Therefore, we can use a combination method which is called F1 (or F-score) to measure. Below showed the concept of TP, FP, FN and TN.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Thus, the formulas of Precision, Recall and F1 score:

Precision: P = TP / TP + FP

Recall: R = TP / TP + FN

F1 score: F1 = 2 * P * R / (P + R)

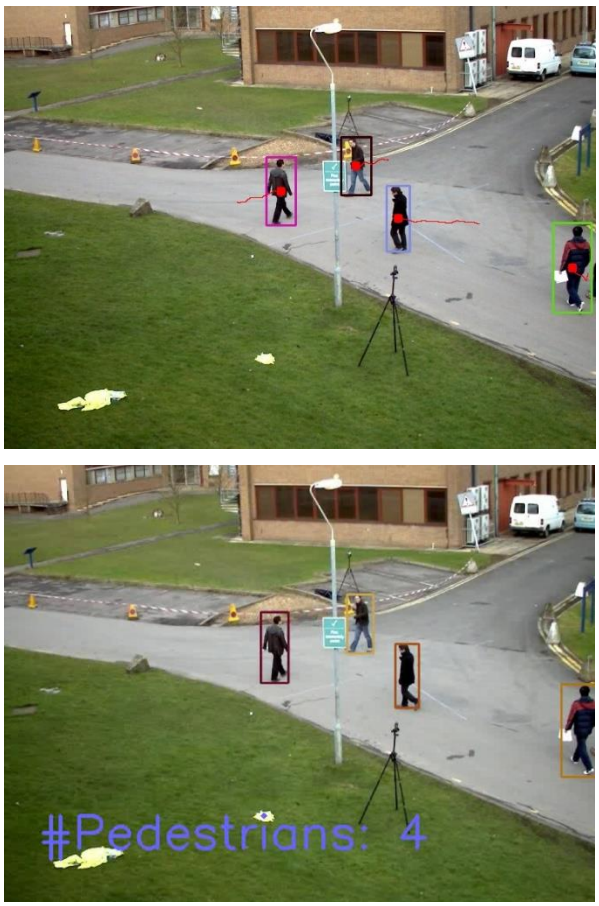
In this project, if there is a bounding box in ground truth but the tracking algorithm does not track, it is FN (False Negative), which means positive objects classified negative. In the same way, if there is a bounding box in the tracking but the ground truth does not have, it is regarded as FP (False Positive) which means negative objects classified positive. Finally, the correctly classified bounding boxes (pedestrians) are TP (True Positive) which means positive objects classified positive.

Task 3:

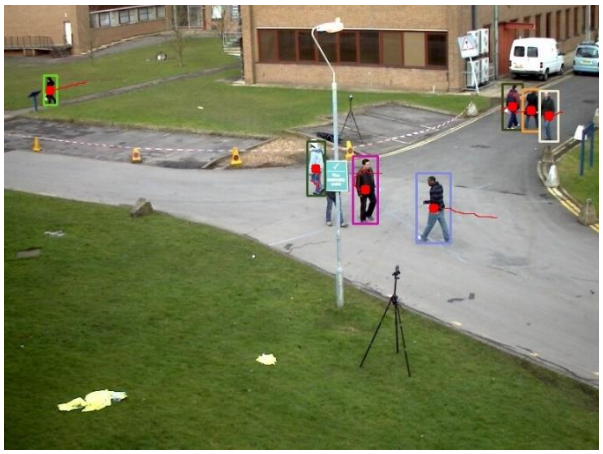
Since task3 is not the same as the previous two tasks, even with ground truth, similar evaluations cannot be used. When we did task3, we found that the result of task3 will be affected by the distance between the pedestrian and the camera. That means the pedestrian's bounding box which is farther away from the camera's perspective will be smaller, which will cause errors in the calculation of the group. Therefore, we only use a simple subjective judgment method here: it is to directly judge the performance of our algorithm from the generated frame results.

V. Results and Discussion

For Task 1 as shown in the picture below, the pre-trained YOLOv3 works good as detecting and maintains a very high accuracy. The pre-trained YOLOv3 is applied to generate the Task1 result. The real time counting on the image is written by Opencv. We used ground truth to validate the results where the precision is 90.4% and F1 value is 90.1%. The recall value and accuracy are respectively 89.8% and 89.3%.

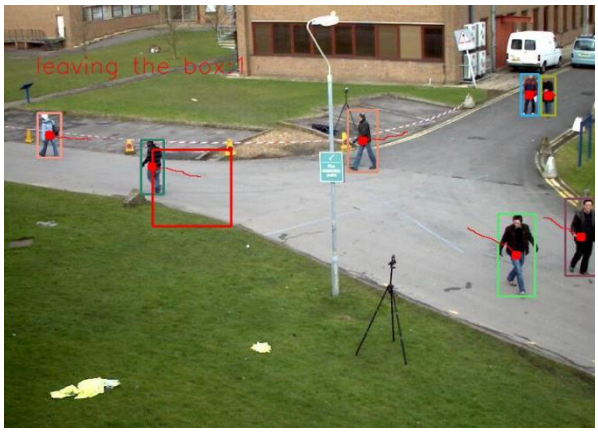


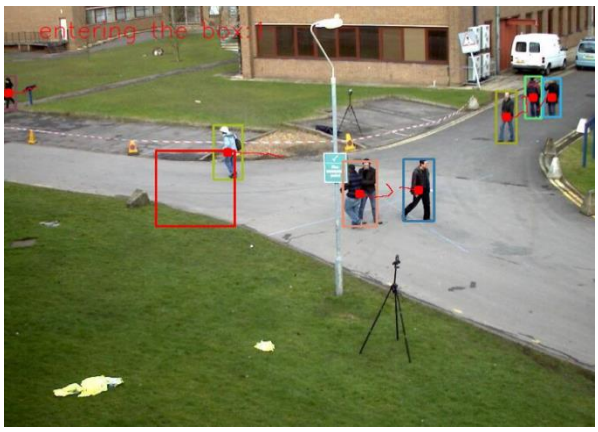
The occlusion is main challenge for our algorithm and there are some occlusion that we can't detect or track:



For Task 2 we need to draw the trajectory of pedestrian and print the real time counts of pedestrian who enter or leave the bounding box. As the pictures shown below, the centroids tracking leads to a satisfactory result. The results are validated against ground truth results. If the difference between our results and ground truth box is less than 10 then the tracking

is considered as correct. By this evaluation the precision comes to 89.4% and F1 value is 89.8% where is an acceptable result. For the accuracy and recall maintained at 89.8% and 90.1% approximately which is a satisfactory result.





Task 3 is about detect a certain pedestrian group. For this task open-cv is used to print the number of groups on image. The group formation is defined by calculating the vectors from frame t to frame t+20, and a group is formed at t if the degree between two vectors is less than 45. Because these is not a validation data set so the validity is done by human eye observation. By observe several sections of videos we found it gives a relatively accurate result of detection. Some result is shown below:

VI. Conclusion

For task 1, this model works perfectly with detecting and draw the bounding box for each of the pedestrians. It realized extracting the number of all moving pedestrians since the start of video. However when pedestrian are sheltered from objects, this model is unable to detect the pedestrian clearly.

For task 2, by implement centroid tracking with opencv it gives satisfactory result on both drawing the path for each moving pedestrians and counting the number of them, by using pre-trained YOLOv3 it generated stable and accurate outputs.

For task 3, by using some geometric theory it could detect the numbers of both independent pedestrians and pedestrian groups. However it doesn't perform well when there is some obstacles covering the pedestrians. It happens because of the obstacles influenced the detecting for our model. So for the future work could be try some other models which are complex enough for this task. And the size of learning dataset is relatively small. So it may need more data for training the model.

Overall, the performance of these three tasks is quite good, but there are still some problems that need to be overcome (such as the occlusion problems). We find that YOLO detection and deep sort tracking can overcome some occlusion, but this method is hard to implement and greatly affected by the version. Many pre-trained models cannot run normally in the new version of Tensorflow or Pytorch. Thus, we have failed many attempts to implement, and the deep sort method is not as easy to understand instead of centroid tracking. Thus, in the future, we can do more research on deep sort to improve our performance of algorithm.

VII. Contribution of Group Members

WENXUN PENG:

Implement Task2 and Task3 and reports

LI XU:

Implement Task1 and Task3 and reports

ZIHAO JIANG:

Implement Task1 and Task3 and reports

PEIBO LI:

Implement Task2 and Task3 and reports

XING XING:

Implement report and task 1

REFERENCES

[1] P.Dollar, C.Wojek, B.Schiele, and P.Perona. Pedestrian detection: An evaluation of the state of the art. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(4):743–761, April 2012.

[2] Fanta Camara, Nicola Bellotto, Serhan Cosar, Dimitris Nathanal, Matthias Althoff, Jingyuan Wu, Johannes Ruenz, Andre Dietrich, Charles W.Fox, Pedestrian Models for Autonomous Driving Part I: low level models, from sensing to tracking. ArXiv 2020.

[3] Ferryman, J. & Shahrokni, A. PETS2009: Dataset and challenge. In 11th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS), 2009.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587, June 2014.

[5] Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.

[6] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. ArXiv, abs/1804.02767.