Multi-Object Tracking through Deep Learning Methods
— Final Report —

Jingyi Fan, Ke Ding, Yipengjing Sun, Yu Hong, Yuanping Qin, Zhaojiang Liu
{jingyi.fan20, ke.ding20, yipengjing.sun20, yu.hong20,
yuanping.qin20, zhaojiang.liu20}@imperial.ac.uk

*all members agree on equal contribution
Supervisor: Dr Francesco Belardinelli
Course: COMP70048, Imperial College London

28th April, 2021

Contents

1 Introduction

Multi-object tracking (MOT), which aims to estimate trajectories of multiple objects in videos, has been a famous problem in traditional computer vision for decades. Tracking algorithms have been applied in a wide range of fields, including traffic control, medical care, and so on. First published in April 2020, the latest version of a new algorithm called FairMOT [20] was updated on the 9th September 2020. It jointly accomplishes the object detection and re-identification in a single network, and largely optimises the levels of detection and tracking accuracy. As an extension work of FairMOT, multi-class multi-object tracking (MCMOT) algorithm enables multi-class tracking. Therefore, in this industrial group project, we will cooperate with Cord Technologies Limited[1] (Cord.tech), develop and evaluate a full pipeline for training and running a new dataset of multi-class multiple-object tracking (MCMOT) problems based on this newly published algorithm.

2 Specification

2.1 Requirements of the project

Based on the FairMOT algorithm, this group project aims to develop a full pipeline for training and running a deep-learning-based multi-class multi-object tracking solution. To achieve the minimum specification, group members should set up a fully functional pipeline which can take in a new dataset with videos on labels, run training over this dataset and run inference of the FairMOT against some datasets with this training model. For higher level requirements, the group needs to write and run evaluation programs that gauge this model against object tracking algorithms found in OpenCV in both accuracy and running time and then optimise the performance of the model against benchmarks. In regard to the extra optional task of this project, the final product should be an optimised pipeline integrated into a visualization platform, which is able to track multiple objects in a video, and its core model can be replaced by another Multi-Object Tracking model.

2.2 User stories

1. Users would like to improve the reproducibility of the algorithms in FairMOT so that the datasets provided by Cord Technologies Limited can be applied to this algorithm.
2. Users would like to get a fully operational pipeline so that they can run multiple-object tracking with their own data.
3. Users would like to integrate the FairMOT model into a platform so that they can directly visualize the results of tracking.
4. Users would like to optimise the FairMOT model with a range of potential datasets so that they can get a better tracking performance.
5. Users would like to compare the performance of FairMOT model with other existing MOT methods such as CSRT and MOSSE in OpenCV so that they can find out the best MOT method.
6. Users would like to get a pipeline in Docker format so that they can deliver the product quickly and easily.
7. Users would like to apply an application programming interface (API) to the pipeline so that they can manipulate it in a user-friendly way.
8. Users would like to run the above steps with another MOT model so that they can have more choices for tracking.

[1] Cord Technologies Limited, 86-90 Paul Street, 3rd Floor, London, EC2A 4NE, United Kingdom

3 Literature Review

3.1 FairMOT

Most of the existing multi-object tracking (MOT) methods can be classified into two categories, Two-Step MOT Methods and One-Shot MOT Methods. Two-Step MOT is the method that object detection is carried out first, then resize the detected target to a fixed size, and extract the re-identification (re-ID) feature. The advantage of this method is that the target scale is well normalized while the problem is also obvious that there is no feature reuse between object detection and re-ID feature extraction, and costs high calculation. As for the One-shot MOT, the target detection also carries out re-ID feature extraction at the same time. The existing methods, such as track-RCNN [14] and JDE [17] (Towards Real-time Multi-Object Tracking), directly add the output of re-ID feature vector in parallel at the detection end of Mask R-CNN and YOLOV3. However, this type of method has the problem of incorrect associated target ID although it can save calculation time.

In this case, Zhang et al. [20] presented a novel approach named FairMOT which surpasses existing state-of-the-art methods in detection and tracking. After they studied the reasons for the decline in accuracy when performing inferences in real-time and there are a large number of objects by several other methods [14, 17], they proposed an approach, namely performing object detection and re-identification (re-ID) equally instead of first detecting and then re-identifying. Qualitatively, this method can perform well in crowded scenes. When the target is severely occluded, the method can maintain the correct identity and bounding box at the same time.

As demonstrated above, object detection and re-ID extraction two tasks are predicted by two homogeneous branches [20]. Specifically, on one hand, unlike other anchor-based methods, the detection task is implemented by elimination of anchors, and represented by a position-aware map via estimating the center and size of the object. On the other hand, the re-ID branch can recognize each pixel-centered object because each pixel is estimated by one re-ID feature. In this circumstance, FairMOT can achieve a balance between these two tasks, instead of the different weights of the detect and re-ID in the other methods.

When it comes to training and tracking, the authors of FairMOT proposed a weakly self-supervised learning method to train FairMOT while implementing the association of bounding boxes by the standard tracking algorithm. For the training, to increase the accuracy, they treated each target instance in the dataset as a separate class, and the objects after transformation are regarded as the same instances in that class. In terms of tracking, the bounding boxes are connected with the existing tracklets by using the cosine distance calculated on the Re-ID feature and their bi-directional matching box coincidence degree based on the initialization of the track of the estimated bounding boxes in the first frame. Note that they used adaptive correlation filters [2] and kernelized filters [5] to update the changes in appearance of the objects.

Additionally, FairMOT may apply on different architectures as a backbone to balance between accuracy and calculation speed and accommodate different types of objects, such as DLA-34, HRNet and so forth. The object detection branch of FairMOT is based on CenterNet [21], so the backbone defaults to use deep aggregation network DLA-34 using deformable convolution in up-sampling modules. This structure can iteratively integrate the characteristic information of network structure, which enables the network to have higher accuracy and fewer parameters. As for HRNet [16] (High Resolution Network), it can also provide fair features for detection and re-ID. Specifically, it has a different structure that changes the connection between high and low resolutions from series to parallel. And it maintains a high resolution representation throughout the entire network structure. What is more, HRNet introduces interaction in high and low resolutions to improve the model performance.

3.1.1 Related Concept of Evaluation

Apart from checking the model performance by directly looking at the tracked images, we applied the built-in evaluation metric in the original FairMOT algorithm to further check the model performance. However, we've found that although the images seem well tracked the performance metric shows imperfect results. An example of the performance metric is shown Figure below10. The first problem

is that though the image tracking results(Figure 2a) seems fine, the metric results show that there is less successful tracked trajectory than expected. Secondly, the number of false detection(FP, FN) seems strangely large and some evaluation parameters' behaviour(MOTA, MOTP) seems strange as well.

To solve this problem, we researched the definition and the usage of the parameters in the metric by literature review. A list of the important evaluation parameters usually used in the MOT challenges are shown and explained below:

1. Overlap: Though overlap is not shown directly in the metric, it is an important basic concept in evaluating the bounding box accuracy. There are two kinds of overlap, spatial overlap and temporal overlap. They are defined as the similarity ratio between the system tracking trajectory and the ground truth(GT) on space and time domain [6].

2. Number of true positive, false positive & false negative(TP, FP & FN): A tracking trajectory will be recognized as true positive (TP) if both its spatial overlap and the temporal overlap is high [18]. After the matching, any remaining unmatched system detections will become false positives (FP) and any remaining unmatched ground truth labels will become false negatives (FN) [6]. These two values will usually be simultaneously considered with other parameters in the evaluation process.

3. Mostly tracked, Partially tracked, Mostly lost(MT, PT & ML): For a pair of system tracking and GT, a closeness value can be estimated by evaluating their temporal overlap. The higher the closeness, the better the model performance [18]. For this particular FairMOT algorithm, a closeness above 80% will be considered as mostly tracked while a closeness less than 20% will be considered as mostly lost [11].

4. Number of switches(IDSW): ID switches indicate the number of times that the ID assigned to the GT by the matched system tracking changes. ID switches may be recognized as TP but it is a wrong detection [18].

5. Number of fragmentation(frag): The fragmentation shows that for a single GT trajectory whether there continuously exists matching system trackings. Ideally, there should be no fragmentation error which means that the tracking system should be able to track continuously and stably for the ground truth object [18].

6. MOTA: $1 - \frac{\sum(FN+FP+IDSW)}{\sum GT} \in (-\infty, 1]$ Scoring function MOTA gives an intuitive measure of the tracker's performance in whether successfully detecting objects and keeping tracks. Three types of tracking errors are measured comprehensively. The detection errors of FNs and FPs, as well as the association error of IDSW [6].

7. MOTP: $\frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}$ Multiple object tracking precision (MOTP) is the total error in estimated position for matched object-hypothesis pairs over all frames, averaged by the total number of matches made. It shows the ability of the tracker to estimate precise object positions, independent of its skill at recognizing object configurations, keeping consistent trajectories, and so forth [20].

MOTA and MOTP are deemed to be the most important metrics for our project's use cases of tracking multiple objects of different types. This is because they give a balanced representation over several parameters. Besides, MOTA and MOTP are complementary in that MOTA measures the classification performance, while MOTP quantitatively reflects the tracking accuracy in terms of the bounding box relative distance to the label. They considered the different aspects of the model performance. What's more, for MOTA and MOTP, a system tracking detection has the same format as the GT. In each frame, it contains a set of estimated detections, each assigned an estimated id, which makes the estimated id from the same detection trajectory consistently unique over time [6]. After detailed comparison, we've found that metric is capable of evaluating actual performance which will be explained in the section below.

3.1.2 Evaluation implementation

For evaluating the models performances, the official evaluation metric for multi-object tracking challenge which is also used by the original FairMOT is our final choice of the evaluation standard. As introduced above, a strange difference between the evaluation metric results and the actual model performances from the tracked videos was encountered during our evaluating process. To solve this problem, we had tried to solve the problems causing the strange behaviour of metric as well as to create our own metric to accurately test the model performance. In conclusion, we have managed to complete both of these two solutions.

For our new metric, we calculated the overlap between the ground truth and the tracking results bounding boxes based on the definition from the references. Then the calculated overlap ratios are used to calculate the basic evaluation scores which are the accuracy and the number of TP, FP and FN in our custom metric. However, although our new metric has better performance than the original MOT metric, it still can't perfectly fit the actual model performance. Although this custom metric is not included in the final product, the function of calculating the spatial overlap is applied in our evaluator class to assist the multi-class evaluation and creating this new metric helped us get a better understanding on the original MOT metric. What's more, it is during the new metric generating process that we found the problem that obstructed the MOT metric from getting correct model performance.

During the trail of creating our custom metric, we've found that the problem causing the strange behaviour of the MOT metric is that the number of generated tracking frames does not match the number of frames in ground truth labels. In other words, the number of generated images does not match the original source image number. This problem is caused by the imperfect of the default image generating package of fairMOT which is cv2. In order to generate the correct number of frames, moviepy is implemented in replacement of the basic cv2 function with our improvement so that the number of generated images correctly match the corresponding ground truth now. This problem will be explained in more details in the technical problems section below. After this problem is solved, it is found that the MOT metric can successfully indicate the exact model performance as well as our custom metric. After comparison, the MOT metric had beaten our custom metric to be the final selection for the evaluations of our models

As introduced above, MOT metric has lots of model evaluating scores. Among all of these scores, MOTA and IDF1 are chosen as our first priority and the ratio of number of mostly tracked plus number of partially tracked over the number of unique objects, which is named as successful detection rate, is set as our second priority. All the other scores will be considered as third priority in our evaluation process. This ranking is determined based on their comprehensiveness and project requirements. The MOTA and IDF1 are the first priority because they are calculated from other evaluating scores and thus they represent the overall performance of the model. MOTA is calculated from FN, FP and IDSW thus it can reflect the overall model performance on obtaining correct bounding box positions[6]. On the other hand, IDF1 is calculated by the ratio of correctly identified detection over the total number of ground truth and detections[6]. Because of that, it reflects the weighted model performance on the re-ID branch. As for the second priority, this ratio straightforward shows how much objects are correctly tracked and thus represent the general model capability. All the other evaluation scores are marked as thor priority but this does not mean that they are not important. On the contrary, since they all reflect the model performance on particular aspects, a good model should be able to have a balanced score among all these parameters.

In summary, in our consideration, a good model should have a high MOTA and IDF1 score since they represent the overall model performance on detection and re-ID branch of multi-object tracking. Also, the more objects successfully tracked, the better the model. Finally, the model should not be particularly weak in certain aspects.

3.2 OpenCV

OpenCV (Open Source Computer Vision Library: http://opencv.org) is a well-known library, which integrates more than 2500 optimised computer vision algorithms [4]. It was originally developed by

Intel in 2000. OpenCV is written in C++, but there are also bindings in Python. The library is licensed with open-source BSD license, and thus can be used for academic and commercial applications. There are 8 algorithms for object tracking in OpenCV. But we will only focus on 2 of them in this project: MOSSE and CSRT.

Before discussing these 2 algorithms, we need to first understand what is correlation filter and correlation filter method. Correlation filter refers to the function to compute correlation for signals [15]. The more similar two signals are, the higher the value of correlation is. Therefore, correlation filter is specifically used to produce sharp peaks in the correlation output. Researchers apply this concept to object tracking and come up with the idea of correlation filter method. Before tracking, the target is initially selected by a bounding box centered on the object in the first frame. From this point on, tracking and filter training work together. The target is tracked by correlating the filter over a search window in next frame; the location corresponding to the maximum value (peak) in the correlation output indicates the new position of the target. Finally, an online update is performed based on that new location [2]. Both MOSSE and CSRT belong to correlation filter method. The main difference between them is the template of correlation filter and the way of update.

MOSSE (Minimum Output Sum of Squared Error) algorithm [2] was published in 2010. It was the first time that the correlation filter method was introduced into the object tracking field [8]. It is a ASEF-like (Average of Synthetic Exact Filter) filter, but overcomes the potential overfitting problem [10]. The training set is constructed using random affine transformations to generate eight small perturbation of the tracking window in the initial frame. Training outputs are also generated with their peaks corresponding to the target center. MOSSE then trains its filter on them. The sum of squared error between the actual output of the convolution and the desired output of convolution is considered as the lost function, and MOSSE minimises it to get a robust convolution filter which can find out the most possible location of the target. MOSSE is more robust to deal with the problem of lighting, scaling, pose, and non-rigid deformations with a fast speed.

Let f_i and g_i a set of training images and training outputs, h the filter. The upper case variables F_i, G_i and the filter H_i are defined to be the Fourier transform of their lower case counterparts. * indicates the complex conjugate and η is learning rate. The filter learned from Frame i is computed as:

$$H_i^* = \eta \frac{G_i \bigodot F_i^*}{F_i \bigodot F_i^*} + (1-\eta))H_{i-1}^*$$

and the MOSSE filter as:

$$H_i^* = \frac{A_i}{B_i}$$

$$A_i = \eta G_i \bigodot F_i^* + (1-\eta)A_{i-1}^*$$

$$B_i = \eta F_i \bigodot F_i^* + (1-\eta)B_{i-1}^*$$

CSRT (Channel and Spatial Reliability Tracker) algorithm [9] was published in 2017. It is much more complex than that of MOSSE. CSRT is based on the Discriminative Correlation Filter (DCF) algorithm. It introduces spatial and channel reliability to improve the performance of DCF tracker. Spatial reliability map which adapts the convolution filter support to a more suitable training region, or in other words, the optimal filter size. This ability excludes unrealistic samples and thus makes the CSRT tracker perform better than the traditional DCF algorithm. Another benefit from the spatial reliability map is its ability to handle non-rectangular targets [10]. Channel reliability is used to weigh the per-channel filter responses in localization, and combine them to get the final response map. It is normalised by the dot product of two values: channel learning reliability and channel detection reliability. The former is calculated during the filter training step, while the latter is computed in the target localization stage. These 2 reliability values play a vital role in the tracking iteration. Detailed pseudo-codes can be found in the paper [9].

4 Design

4.1 Overall Design

The users can run our pipeline through either API or CLI to perform multi-object tracking. After setting up the environment on google cloud platform, the users need to input an API key and project name to generate the dataset information first, and then they can access our product pipeline. The subsequent workflow is shown in the Figure 1 and has two main structures.

In Figure 1, the red words indicate the information that users need to interact on the API. '(C)' and '(I)' represent the format of user input on API. '(C)' represents choice; '(I)' represents manual input. The light blue words are the default value of each input. The diamond represents the result returned to the user on the API.

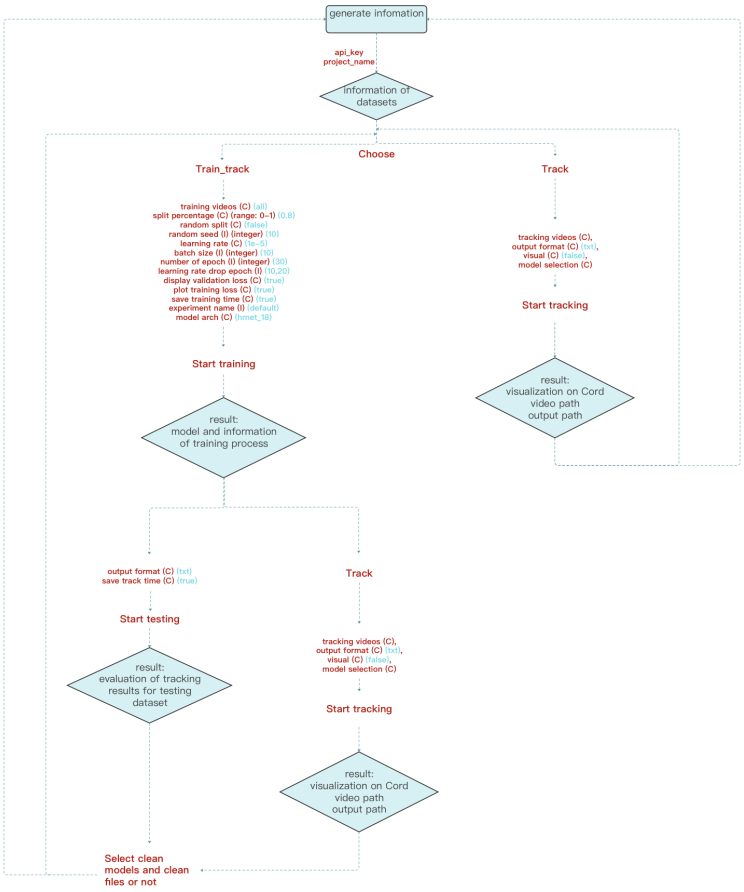


Figure 1: API flow chart

4.1.1 Train-track Procedure

4.1.1.1 Train and Test for Model Selection

When the users input the split percentage and other information, the pipeline would divide the user-specified videos into two parts, training and testing, based on the split percentage entered. Then the pipeline would generate a model trained on the training dataset. After that, multi-object tracking will be performed on the testing dataset and the evaluation results would be obtained. It is worth mentioning that the users can use this function multiple times by different split percentages to generate different models and compare their performances on the corresponding testing datasets. Through these evaluation results, the users can get different performance of FairMOT algorithm on variance training dataset scales.

4.1.1.2 Train and Track the Whole Video

The function of training and tracking the whole video is designed to be very similar to the previous one in the steps of splitting the video and training the generated model. After that, the user would returns to the previous page. In this case, they can click 'track' and select a model to do the multi-object tracking on the specified video. In addition to using their own trained models, the users can also choose the pre-trained model that we provided. Depending on the users' choices, the pipeline can return visual video results, which will be displayed on the Cord platform.

4.1.2 Directly Track Procedure

When it comes to the function of tracking videos, the users can directly select the prepared model provided by us to perform multi-object tracking on their own videos. This is a relatively convenient function, and the users can visualise the results on the Cord platform same as the visualisation of the training and tracking the whole video function.

4.2 Other Designs Considered

4.2.1 Evaluation of the Whole Videos

In our initial design, we considered to provide users with a choice whether to evaluate the whole video rather than just the testing part after they have trained on their own datasets. However, after discussion, we thought that the evaluation of the tracking results should not be used for the whole video, because the whole video contains the training set and the evaluation results only on the testing set can not reflect the performance of the model. In this case, if the users want to visualise the whole results of the video after the training, they can follow the instructions in 4.1.1.2.

4.2.2 Train-Test Split Strategy

For the method of dividing the datasets into training and testing sets, we initially made the following design.

4.2.2.1 Inter-video Training and Testing

If the users' projects contain multiple videos with labels, the user can select several of them for training and the rest for testing.

Our pipeline can achieve this split strategy. Users can select the videos they want to train and test, and specify the split percentage corresponding to the videos used for training as 1 while the split percentage corresponding to the videos used for testing as 0.

4.2.2.2 Intra-video Training and Strategy without Random Shuffled

Additionally, each user-specified video will be split into the part of training and testing according to the corresponding split percentage as described above, the split points were designed to be random and can be controlled by 'random split' choice. If the user wants to randomly split, then a part of the video will be intercepted as the training part, and the remaining sets will be used as independent testing parts. We took this strategy as our design because the frames in training and testing parts are sequential and can be seen as new videos. It is more easily to use our evaluation.

4.2.2.3 Intra-video Training and Strategy with Random Shuffled

In our initial design, we considered randomly shuffling all the frames in each video, and then split frames into training and testing parts. In this case, since the frames of the test part are not sequential, it is necessary to track the entire video to include all frames in the testing sets. What is more, the processing time is too long and the evaluation needs to be redesigned, so we didn't adopt this design in the end.

5 Methodology

5.1 Plan

The user stories in part 2.2 can be divided into seven sub-problems: Parser (1), Pipeline (2), Visualisation (3), Maximizing performance (4), Benchmarking (5), Dockerization (6), API (7), Extra task (8). Based on their priority, necessity, feasibility and workload, we worked out the following plan:

Task	Estimated duration	Priority	Necessity	Feasibility
Parser (1)	Long	Very high	Mandatory	High
Pipeline (2)	Very long	Very high	Mandatory	Normal
Visualization (3)	Short	High	Mandatory	High
Maximizing performance (4)	Long	High	Optional	Low
Benchmarking (5)	Short	High	Mandatory	High
Dockerisation (6)	Short	Low	Mandatory	High
API (7)	Short	Low	Mandatory	Normal
Optional task (8)	Short	Low	Optional	Normal
Final report, presentation	Long	Low	Mandatory	High

Table 1: Work Plan of the Project

Story 1 is a kind of preprocessing. Without it, the input data cannot be compatible with the FairMOT algorithm. Therefore, the priority of the story 1 should be the highest and it is mandatory to accomplish it at first. Then, story 2 aims to construct and train the model. This is the core part of the project and thus mandatory. Next, story 3 means visualization. We can observe the results of tracking more quickly and easily, and it is obviously easier than optimizing the model. Therefore, the priority of story 3 is higher than that of story 4. Visualization is mandatory, but optimization is optional due to its difficulty. There is no doubt that story 5 is mandatory because this step measures the performance of the final product. Story 6 aims to use Docker to deliver all the code in a package. This mandatory process is a kind of conclusion and can only be done at the end. Story 7 is a kind of manipulation simplification for users. The API needs to be connected with the Docker image. As for the final story 8, it is an extra optional task which doesn't belong to the main body of this group project. That is only for further research.

As for the feasibility, story 1, 3, 6 have a high feasibility. Story 1 is only a transformation of one format into another format. Story 3 and 6 will directly use the third-party software Docker and the platform provided by cord.tech. Story 5 evaluates the performance of the model. It is not difficult after finding a proper metric. Techniques involved in these stories are mature and group members have already learnt them before. On the contrary, the feasibility of story 2, 4 and 7 is not so high. The first two ones are closely related to the FairMOT algorithm. This algorithm has already been a state-of-the-art algorithm. It is hard to find a way to improve its performance in a short time. In addition, whether the most suitable parameters of the model can be found or not depends on the size of the training dataset and the training time. Therefore, the difficulty of mastering this new technique and time cost should be taken into account. Story 7 requires the knowledge of API design which needs some time to learn.

5.2 Techniques

When it comes to the techniques we would use in this project, we met a lot of technical problems that have to be solved. For example, we will use the language Python throughout the project, including the library Py-torch, OpenCv and so forth. For the API part of the project, we've used the Python flask package as our backend and we've created several HTML pages as our frontend. In addition, Postman is used to assisting to pin the frontend. Since the project is a method related to deep learning and the model needs a large computation, we will work with Google Colaboratory and Google Cloud Platform which provides GPU support. Specifically, Google Colaboratory is used for python language

while we can build a docker container in Google Cloud Platform. It is worth noting that we must first configure the environment to make the version of the toolbox and environment compatible with each other when we use these two platforms, such as the version of Miniconda, GPU Driver, Cuda and so forth. Moreover, we created a group Google account, so that we can upload the relevant datasets to Google Drive and work on the project together. Besides, Slack is a good software so that we can contact not only each other but also the product owners with higher frequency.

As for the version control system, we use git and organize our project on Gitlab so that we can work together and manage code easily, including establishing branches and tracking the historical changes. What is more, it would save a lot of time especially when we get familiar with the git commands.

5.3 Technical Problems

5.3.1 Single-class to Muti-class Muti-object Tracking

The original version on Github repository of the FairMOT [19] is for single-class. Even the dataset contain multi-class objects and muti-class labels, the algorithm can only track one class of objects. After diving into codes in the repository and browsing related work and questions about FairMOT carefully, we found a extension work, MCMOT by CaptainEven [3]. It extends the one-class multi-object tracking to multi-class multi-object tracking. For the reason that MCMOT highly conforms to the requirements of our client, we set up our pipeline based on this extension work.

5.3.1.1 Training Process Difference

The main difference during the training process between the MCMOT and the FairMOT is that MCMOT sums re-ID loss of all classes of objects during the training process. Then four kinds of optimization choice for re-ID loss are provided, cross-entropy loss, circle loss, GHMC loss [7] and cross-entropy plus triplet loss [3].

5.3.1.2 MCMOT in Detection and Tracking

The flow of detection and tracking for MCMOT is shown in Figure 2 [12]. Different from FairMOT, MCMOT passes each class of label into the neural network for detection. After detection, for each class, the re-identification embedding of each class will be correlated to the detection based on that class, rather than associating it to all classes. After re-identification, the MCMOT uses Kalman Filter and Hungarian Algorithm for tracking.

Kalman Filter is used to predict the position of object in next frame according to the its position in the current frame. Kalman Filter contains prediction stage and update stage. Prediction stage is predicting the position of the object in next frame. Update stage is updating the position of the object based on the observation of the prediction and the ground truth.

The prediction stage involves two equations:

$$\begin{aligned} \hat{x}_{t|t-1} &= F_t \hat{x}_{t-1|t-1} + B_t u_t \\ \hat{P}_{t|t-1} &= F_t P_{t-1|t_{-1}} F_t^T + Q_t \end{aligned} \tag{1}$$

where $\hat{x}_{t|t-1}$ is projection of state ahead and $\hat{P}_{t|t-1}$ is projected covariance ahead.

The update stage involves two equations:

$$\begin{aligned} \hat{x}_{t|t} &= \hat{x}_{t|t-1} + K_t \left(z_t - H_t \hat{x}_{t|t-1} \right) \\ \hat{P}_{t|t} &= \hat{P}_{t|t-1} - K_t H_t P_{t|t-1} \end{aligned} \tag{2}$$

where $\hat{x}_{t|t}$ is updated estimate of state based on measurement and $\hat{P}_{t|t}$ is updated error covariance.

Hungarian Algorithm is an a combinatorial optimization algorithm that solves the assignment problem. In MCMOT, it is used with a cost matrix against distance to match the objects in next frame to the current frame.

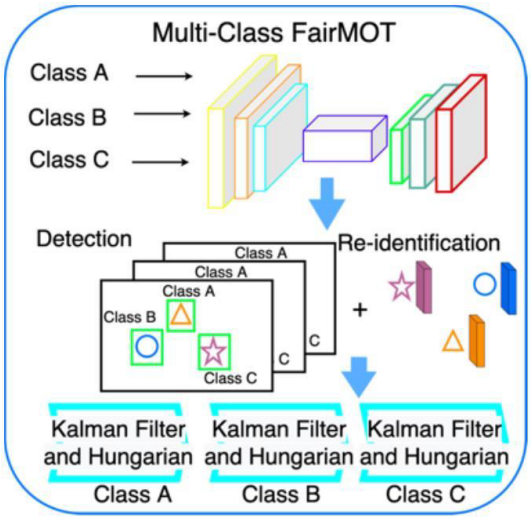


Figure 2: Illustration for MCMOT detection and tracking process

5.3.2 Early problems with FairMOT evaluation

It turned out that adapting the original FairMOT evaluation scheme to accurately reflect model performance (i.e. in agreement with the visual examination of the tracked output video)was quite a tremendous challenge in earlier stages of this project. In fact, details about the problems encountered are discussed in depth in Report 2. Some of the biggest problems include but not limited to: the evaluator detects no successful objects being classified as tracked that is any object satisfying either mostly tracked (MT) or partially tracked (PT) conditions, despite of the fact the visual output shows that the model did place most bounding boxes at correct coordinates for each object. Another problem is the number of false positives (FP) labelled by the evaluator is incredibly high (almost as much as there are the total amount of objects in the whole video).

Tasks of dealing with such a severe issue were given the highest priority. We came up with two possible options: either to modify and fix FairMOT evaluations or devise our own version of MOT metrics and evaluation. We decided to stick with the built-in evaluation mechanism of FairMOT, considering the benefit of using State-of-the-Art MOT object tracking metrics and the efficiency of FairMOT's built-in style implementations.

And after a lengthy period of testing and debugging, we successfully located the cause of the problem to be within our initial dataloaders. Specifically, we found out that the problem is caused by the `OpenCV`'s video parsing methods we initially adopted contained inherent uncertainties in parsing the input video into the exact amount of frames at a specified frame rate. This uncertainty in the number of frames produced by the `OpenCV` method is direct cause for a mismatch between the number of input raw frames into the model and the number of frames specified by the GT(ground truth) file which acts as a label for the subsequent model training. The number of frames in the GT file is unaffected by our local video parsing methods because the GT file containing the bounding box(bbox) information is directly fetched from Cord platform and it has the original number frames matched with the video's frame rate.

To rectify this mismatch, we opted to use `moviepy`'s `save_frame` methods. Specifically, we constructed the an array in which each element is a checkpoint in the video's time-space, incremented by the time-interval defined by the frame rate accurately read by `moviepy`. This method allows us to generate the exact number of frames from every video as we double-checked each frame number with the frames on Cord. This leads to the same number of raw frames to be passed into the model as there are the number of frames in GT labels, which completely solves the mismatch problem. Then we successfully adapted the FairMOT evaluation to be within our pipeline such that it can reliably reflects the true model performances for different models across different datasets.

5.3.3 Problems with multi-class evaluation

After we overcome the FairMOT problems, it became clear to us that the vanilla FairMOT is unfit to perform multi-class multi-object tracking for our pipeline. And the multi-class tracking capability would be a significant selling point for our pipeline to have(as directed by the project owner Eric), we thus decided to move our pipeline to be based on MCMOT, as with previously discussed, MCMOT is essentially a modified version of FairMOT with modified multi-class loss functions in the training and tracking scheme.

However, even the powerful MCMOT does not support multi-class evaluation as the MCMOT's evaluator is basically copied off from FairMOT and it can only perform binary evaluations (that is for objects of different classes it can only treat all classes as one class and evaluate whether that one class is tracked/classified or not).

5.3.4 Building the true multi-class evaluation

Realising this inherent drawback of MCMOT, we began to engineer ways to equip the existing FairMOT/MCMOT type evaluator with true multi-class capabilities. We first found out that in MCMOT there actually is class id information for each object coming out of the model output but not utilised inside the evaluator. We immediately started constructing a multi-class evaluator(`MCEvaluator`) class that utilises the class information provided by the powerful MCMOT. As a side note, for readers who are interested in our exact implementations of such multi-class evaluator, its source codes can be found by navigating to our repository's main and cd to */src/lib/tracking_utils/evaluation.py* and locate the `MCEvaluator` class inside.

We then conducted research and found the evaluator is nothing but based on the universal MOT challenge metrics package `py-motmetrcis`. We studied in great depth about `py-motmetrcis`'s implementations and realised that the core of the evaluation operation is performed by the `acc` object instantiated from the `MOTAccumulator` class [13]. This accumulator object `acc` is updated every frame and it is updated by passing in the GT bbox info as well as the detected bbox info as a pair for each object. We realised what we actually need is multiple accumulators one for each class, instead of a single accumulator that binarise all classes to one.

With this knowledge, we transformed the problem into creating multiple accumulators and establish the object-class relation for each object at every frame. We first initialised the number of accumulator objects to be the same as the total class number, then stored object-class relation information in dictionaries for both GT labels and detection results. We then carefully cycled through all of the accumulators (all classes) and update each one whenever the class id of the GT object bbox matches. Note here we conditioned the normal update to only take place when that particular class present in GT labels is also present/detected in the detection results. If otherwise we performed what we call the "empty" update where for that particular class id not at all detected we passed in an empty object bbox info(coordinates of bbox) so that the accumulator would know that object is completely missed.

The `MCEvaluator` indeed achieved true multi-class evaluation (one example of the mc-evaluation results can be seen in Figure 10) and it effectively enables our pipeline to be fully multi-class: multi-class training, tracking & evaluation. The other added advantage over the original MCMOT/FairMOT evaluation is it now fits better the definition of MOT metrics by adding the option to narrow down the amount of objects covered by the background/absorbing class, allowing more accurate evaluations [1] i.e. before only one big class is considered and everything else is the absorbing class; now user can specifically define a absorbing class by adding another id to the class id dictionary.

Overall this multi-class evaluation improvement based on MCMOT's existing capability of multi-class training and tracking is one outstanding achievement of this project. We recognise this to be one of the key features that adds value to our final product as its multi-class capability surpasses the original designs.

5.3.5 Technical Problems during Setting up the Environment

Moreover, when we built the docker container on the Google Cloud Platform, we initially used the Container-Optimized OS as the virtual machine which is design for docker by Google Could. However,

we were stuck in setting up the environment, the version of NVIDIA driver is not compatible with the version of the CUDA. In addition, the CUDA version of the docker file in FairMOT is also not compatible with that of the Container-Optimized OS. After constantly seeking solutions, we finally established another virtual machine of Ubuntu system and manually install the Nvidia docker as well as Nvidia driver. Then we can dockerise our pipeline on Google Cloud platform using GPU.

5.4 Software Development Technique

In regard to the development approach, we had a thorough discussion to determine it at the beginning of the project. After discussion, we decided to choose Scrum as the strategy to complete the entire project. Based on Scrum, the project owners are Eric Landau and Ulrik Stig Hansen, who are the industrial partners from cord.tech. Francesco Belardinelli, the supervisor of this project, is a Scrum master who is responsible for supervising, monitoring the progress of the project, ensuring the completion of the project and protecting our team. There are 6 people in our group where Jingyi Fan is the group leader who is in charge of mainly contacting project owners and Scrum master as well as organizing meetings and monitoring progress within the team while other students are members.

For the Scrum flow, through the discussion with product owners, we determined the product and sprint backlog which will be accomplished in the future. Product backlog is mentioned as minimal specification above, and sprint backlog is the plan and schedule described above as well. After each sprint, we would make a review and retrospective within the team and make a sprint planning meeting with product owners to discuss the beginning of the new task. Besides, we would make another two meetings, one is meeting with product owners every two weeks in the spring term and every week in the Easter period, the main content includes communicating with the product owners the task we accomplished in the previous two weeks and a plan to accomplish in the next two weeks (or one week in the Easter period), and raise any questions around the project. These meetings can ensure that the work we are currently doing meets the needs of our customers. If there is any deviation, we can adjust it in time. And another is meeting within the team, the time that we will make a provisional appointment before the meeting because of the time zone and coursework of different team members. The content is similar to the former but for individuals. When we finish the whole project, we will provide a final report of this project and make a presentation with a demo.

5.5 Quality Assurance Methodology

The project requires us to use FairMOT to develop and evaluate a full pipeline to train and run a custom dataset to address multi-object tracking problems. We will demonstrate how we strictly tested the quality to better accomplish the project and achieve the specification described above in this section.

In order to test the model quality, in the beginning, it is worth to mention that the system under test of this project is a deep learning algorithm and we need to test the prediction accuracy i.e. tracking accuracy, of the model to fit for purpose. What's more, as the accuracy is related to the evaluation metrics, we need to reiterate the metrics and measurements we used in the evaluation during the testing illustrated in the above section, such as FP & FN, IDSW & frag, MOTA & MOTP, etc. Generally, compared with benchmarking, the expected outcome is that all moving objects can be detected by bounding-box accurately, and the detected id of the same object will not lost or changed during the tracking(i.e. High MOTA & MOTP and low IDSW & frag).

For the entire project, we need to get our FairMOT model running first and then generate some baseline metrics. In expectation of testing the model quality, we need to take some data and use some subsample data to train the model. After that, we would evaluate the FairMOT model on the rest of data to look at the accuracy. The next part of the project is that we would work on different ways by generating different results of slicing the data once we have made our pipeline running. Particularly, we tested the accuracy of the model when slicing the data in different ways through evaluation metrics. For instance, before each tracking, we should take only every 50 frames of the video to train, and train on every 30 frames, on every 20 frames, 100 frames and so forth. Additionally, In the process of completing the project, it is gradually borne on us that some possible sources, like

data or infrastructure may make the model unfit for the purpose. On one hand, the information of the given data is incomplete, i.e. some values in the ground truth file are uncertain, resulting in problems with the testing performance. On the other hand, since FairMOT is a single-class tracking algorithm, for the multi-class tracking goal of this project, such as pedestrians, trucks and cars, we proposed to employ multiple networks(each tracks one type) and merge predictions frame-wise. This is a change of infrastructure inside the algorithm as the system under test is a white box whose internal codes can be modified.

Moreover, we also related our quality assurance plan to development strategy, Scrum. In accordance with the requirements of the product owners, we completed the setup of the pipeline and made sure the model worked properly, that is performing normal training and tracking according to the FairMOT's authors' instruction. And then we started training and testing the model according to the above ways. After each testing, we would report whether the performance of testing was good in internal meetings within our group and update the outcome in a bi-weekly meeting with the product owners in charge to get some feedback. Furthermore, we would try to generate some baseline metrics and do some benchmarking experiments to make a full comparison with our model. If the accuracy of the testing has a good performance and the product owners also think the test can pass, we will proceed to the next step, namely visualising and dockerising.

6 Experimentation

The original FairMOT is a single-class multi-object tracking algorithm, thus almost all the experiments they have done based on the single class, pedestrian. Nevertheless we chose to use an adapted FairMOT algorithm which considers multi-class information, since we need to deal with the MCMOT tasks. The DLA-34 mainly outperforms other models in the original FairMOT algorithm since the deformable convolution in DLA-34 alleviate the mis-alignment issue caused by down-sampling for small objects which makes the DLA-34 more discriminative on the Re-ID features [20]. However, in our experiment while using the Traffic Detection Dataset which contains five classes, car, bus, motorbike, pedestrian and truck, the performance of DLA-34 was worse than the HRNet-18, DLA-34 could hardly track the pedestrian objects successfully while almost 90% of objects are belonging to class car. The Figure 3 below shows the compare of the DLA-34 and HRNet-18 on one representative example frame. The two models used the same hyperparameters, the learning rate is 1e-5, the number of epochs is 30 and batch size is 3. The batch size is kind of small since otherwise the CUDA would be out of memory in the Colab. One of the reasons that could result in such differences might be the dataset itself. Since the dataset is kind of small and not be fully labeled each frame and there are only eight pedestrians objects and one bus object in the whole dataset which might result in not learning enough feature of pedestrian and bus classes. However, under the same circumstances, HRNet-18 almost tracked all the objects appeared in the ground truth, which shows that in MCMOT tasks the HRNet-18 might outperforms the DLA-34.

(a) hrnet-18 (b) dla-34

Figure 3: Comparison of the DLA-34 and HRNet-18 on one example frame

In the the dataset of Imperial Drone Project that contains only one class, cattle. While experi-

menting on this dataset, DLA-34 outperformed the HRNet-18 because of the discriminative ability of the Re-ID features. The two models were compared with the same hyperparameters as above. The two MOTA scores are both extremely high since the dataset provided only has eight labeled objects. But the differences of the MOTA scores are because of the different Re-ID abilities. After tracking the whole video, the number of ID switches of HRNet-18 is much higher than DLA-34. In general, HRNet is more suitable for the MCMOT tasks, but DLA-34 is more stable while processing the single class tasks.

Model	MOTA
DLA-34	0.984379
HRNet-18	0.94211

Table 2: Comparison of MOTA on the Imperial Drone Project

According to the requirements of clients, one essential task we need to experiment is to compare the performance of the FairMOT algorithm over different datasets split methods. Since in general there will not be sufficient datasets for clients to train a new model, thus it is necessary for them to know whether the performance of training on a small dataset could also reach to a high evaluation score compared to train on a larger dataset.

The dataset we chose in this experiment is the Traffic Detection Project provided by the Cord platform. There are four labeled sequences in total and 2703 frames in the whole dataset. We made nine splits in total, the percentages of the training sequences which were randomly chosen were from 0.1 to 0.9. We chose the HRNet-18 as the model architecture since the HRNet-18 outperforms the DLA-34 for the multi-class objects tracking task from our experiment above. The batch size are three otherwise the CUDA would be out of memory in the Colab. The number of epochs was chosen fifty, then we could prevent the model from overfitting by observing the training and testing losses.

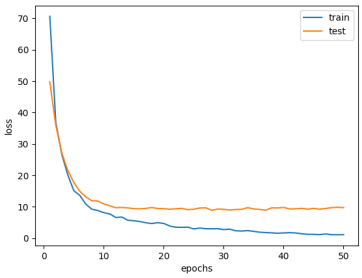
From the Figure 4 and Figure 5 we could find that the different amount of training frames in this experiment would not affect the training performance much. The behaviour of the training losses and test losses are almost the same but with slight differences. After collecting the training results, for each split we picked the best training performance model through fifty epochs and tracked their testing datasets to justify whether the model trained on a small dataset could also reach a high tracking performance.

We selected three main evaluation methods to assess the tracking performance which are MOTA, IDF1 and track ratio. As shown in the Figure 6, the MOTA and IDF1 raise in fluctuations and reach a peak while the train split ratio is 0.8. The two scores finally drop when the ratio increases, which is might because the testing dataset at the 0.9 train split ratio has rather small frames, in a particular sequence, there might exists several objects that are hard to track finally interrupting the tracking evaluation. However, without considering such noises when the train split ratio is too low or too high, the overall trend in the Figure 6 seems reasonable. In this experiment, only if the train split ratio is higher than 0.5, the final tracking score would reach to the acceptable range. Therefore, the final performance of a model trained using the FairMOT algorithm would be significantly affected if the dataset is too small.

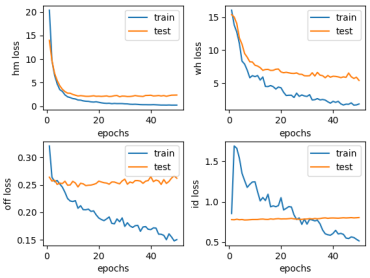
According to the experiments experience, the learning rate should be in the range of 1e-5 to 7e-5 and the number of epochs should in the range of 20 to 30. Such selections of hyperparameters would generalize the model well.

7 Division of Work

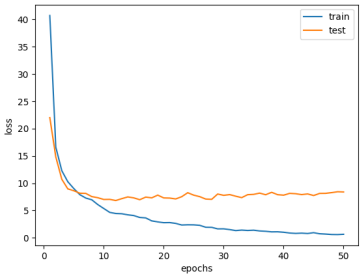
The overall roles played by each group member: Ke Ding plays a central role in decision-making process; Ding Ke and Yu Hong tackled the most technically challenging problems in parser, evaluation



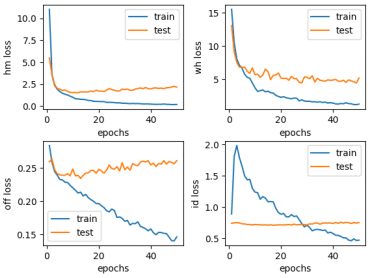
(a) Total loss 0.1 split



(b) Sub losses 0.1 split

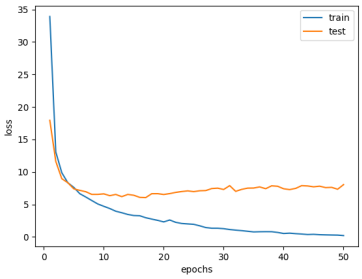


(c) Total loss 0.3 split

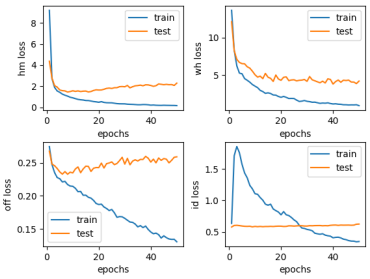


(d) Sub losses 0.3 split

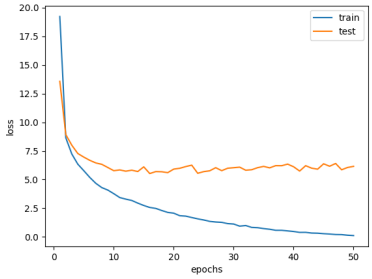
Figure 4: Training and testing losses in one experiment of the Traffic Detection Project when the split percentage is 0.1 and 0.3



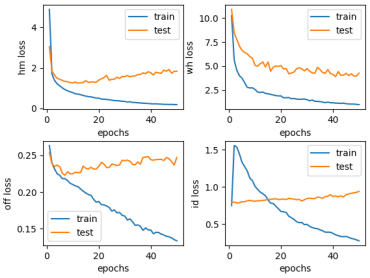
(a) Total loss 0.5 split



(b) Sub losses 0.5 split



(c) Total loss 0.8 split



(d) Sub losses 0.8 split

Figure 5: Training and testing losses in one experiment of the Traffic Detection Project when the split percentage is 0.5 and 0.8

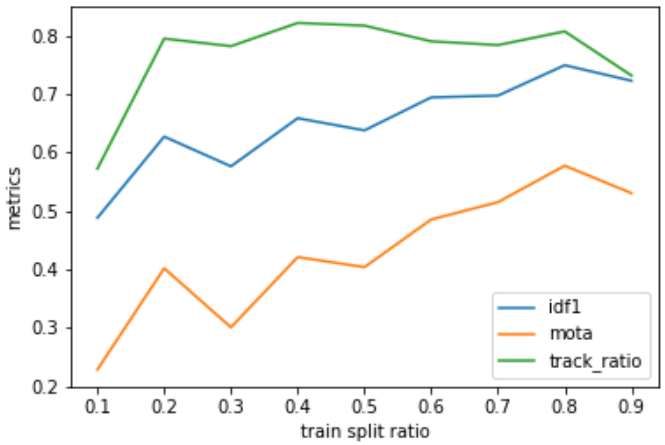


Figure 6: Tracking evaluation over nine splits

and pipeline; YipengJing Sun and Zhaojiang Liu developed and debugged most of the pipeline. Jingyi Fan and Yuanping Qin conducted benchmarking and research for this project. Jingyi Fan also played the role of document editor. It is also worth mentioning that Ke Ding, Yu Hong, YipengJing Sun and Zhaojiang Liu played pivotal roles in advancing the project forward, and that Yuanping Qin and Jingyi Fan supported in doing so.

At the beginning, based on the task of our project, that is, setting up the pipeline for the model to be functional, benchmarking, visualizing, and dockerisation, we provisionally split our group into two subgroups, backend and research, respectively, with 4 members for backend group and 2 members for research group. The task that the research group is responsible for is benchmarking and dockerisation, and the backend group is in charge of the other two tasks.

As mentioned above, the backend group started with setting up the pipeline and getting the parser working. In terms of the parsers, they took the structure of new datasets and then parsed it into the format that would be ingested by the model. After getting the code up and running, they could focus on visualization, that is using the provided software to visualize the data. Research group would find out the metrics that they are going to use. Specifically, they established the metrics that would be based on the benchmarking first. After figuring out what kind of metric we would want to use to gauge the accuracy of some OpenCV trackers (e.g. CSRT, MOSSE), they did some benchmarking experiments with these trackers. Then they would apply to dockerise the whole pipeline.

Furthermore, it is worth mentioning that in order to improve the efficiency, we divided our group into several task groups based on the two subgroups in the Easter period. Each group would be assigned different members according to the difficulty of the task, most of which are 2 people per task group. Specifically, We discussed 3 tasks in the next stage based on the previous project progress and set a short-term deadline within the group meeting, each two members responsible for one and each group must complete the task before the DDL. After completing the tasks in this phase, we discussed within the group again and decided on the next 3 tasks with three new task groups on the basis of the previous 3 tasks. If there is a difficult or complex task, we would appropriately reduce the task groups and assign more members to that difficult task group. Until the tasks are completed, we have established a total of 19 task groups.

8 Final Product

Our final product has two versions: API version and traditional CLI version. These two versions could both achieve main tasks in the specification. Thus, in the overview of overall achievements, API version would be introduced as an example, but both two versions would be fully discussed in the subsections.

8.1 Overview of Overall Achievements

The API version of our final product that has been ultimately designed and produced is a web-based application which could implement the multi-class and multi-object tracking tasks based on FairMOT algorithm. According to the requirements of clients, our product provides two main usage methods which have been shown in the flow chart of Figure 1. The first one that shows in the flow chart called Directly track procedure is suitable for the scenario that clients intend to track a raw video by a pre-trained model that we provide and visualize the tracked video on the Cord platform. If clients would like to train a model on their own dataset, they have to choose the second method which is called Train-track procedure. In this way, clients could train a model using any datasets they want. In addition, if they intend to check the tracking performance of the model that they trained and make model selection, there is a dataset split option so that they could track the test dataset to produce the evaluation table after training their own model. Furthermore, clients are also capable to track videos and visualize them on the Cord platform by their own models. The two main usage methods above are the overview of our final product, the detailed functions of the two versions will be discussed below.

8.2 API Version

8.2.1 API Introduction

Based on the project specification, a set of HTML based API is created for ease of use. The user of our program can easily run through the full pipeline without any command-line operations. Basically, our API has the exact same capability as the command line method. But, compared to the command line operation, all the parameters and settings can be modified on the website much easier. What's more, since we have a step-based HTML design, the user can easily run through our programs with three possible predefined routes by running through certain pages of our API.

One point to note is that we have made several improvements to our API design to create a friendlier environment for the user. Firstly, we've summarized some basic training parameters such as the learning rate and the number of training epochs as basic settings for the training process. The user can either start training simply with these basic settings defined or modify all the training related parameters in advanced settings if wanted. Secondly, unlike typing in the names of the prefered videos and models in the training and tracking process using the command line, the user can simply choose from available options provided on the webpage. Of course, these videos and models information will be automatically updated each time this page is opened. The detailed flow of the API is shown in the sections below.

8.2.2 Generating Information Function

Generating information function is the first part of the product pipeline which is aimed to download videos in the datasets and display the related datasets information. Since this product is designed for the Cord Technologies Limited, thus the only way to access the datasets that clients need is through the Cord API. Therefore, in the main page of our final product, clients have to type in the project name and API key to access their datasets. The videos and seqinfo.ini in their datasets would be promptly downloaded and automatically classified into labelled and unlabelled videos after they submitting project name and API key. Meanwhile, the pre-trained model would also be downloaded at this step in order to allow clients to directly track their raw videos without training. In short, this function is designed for the clients to see the names of videos in the datasets and check if the videos are labelled.

8.2.3 Directly Track Procedure

After generating related information of datasets, users could choose the most convenient way to experience the power of the FairMOT algorithm which is the Directly track procedure. In this procedure, a pre-trained model that has been downloaded in the Generating information function would be provided to users to track their own videos. This pre-trained HRNet model containing five classes which

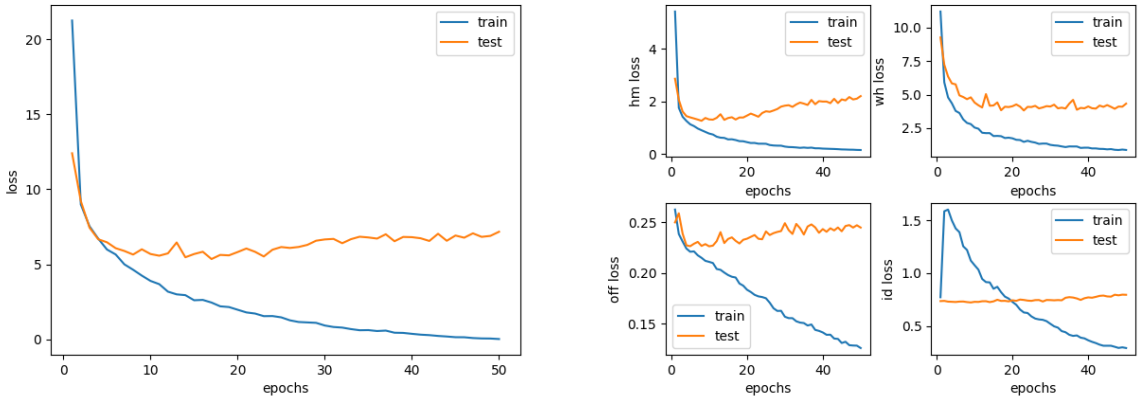
are car, bicycle, person, cyclist and tricycle was trained on the DETRAC-Dataset. Users just need to select their videos they intend to track and choose the visualization option, then the tracked videos would be shown on the Cord platform.

8.2.4 Train-track Procedure

Train-track procedure is made up of three components: dataset preparation, training and tracking. Users have to input or choose some necessary information in the webpage to maintain the procedure's normal operation.

8.2.4.1 Dataset Preparation

For the component of dataset preparation, users need to select training videos among the whole datasets. If they intend to split the dataset into train and test datasets, they could type in the split percentage to decide the percentage of training part. Furthermore, users also need to decide whether split randomly or not, the randomly split method we chose is that we randomly chose a sequence of frames for each training video as the training dataset and other parts of frames as the testing dataset instead of random shuffling, since random shuffling would possibly harm the re-id ability while training. There are two main reasons that we created the split datasets function. The first one is based on the training perspective, if user intend to train a generalized model, they should prevent the model from overfitting. If training dataset is fixed, it could be achieved from hyperparameters tuning. However, our clients' dataset is varying from projects, thus we cannot recommend some specified hyperparameters. After splitting the dataset, users could easily justify whether the model is overfitting or not in the losses plot. For example, the Figure 7 below shows the training and testing losses in one experiment of the Traffic Detection Project when the split percentage is 0.7, which means the training dataset is 70% and the testing dataset is 30%. It is obviously that the testing losses of heat map and offset gradually rise from around twenty epochs, the total testing loss also has the modest growth. Therefore, in this case users could relatively easily train a better model to prevent overfitting. Moreover, the second reason is from the view of tracking evaluation. According to the task specification, tracking evaluation is an essential part, which means that clients need to see and check the tracking evaluation to decide whether they would choose the model they trained. However, if the evaluation of a model skill on the dataset that contains the training data would result in a biased score. Therefore the model should be evaluated on the held-out sample to give an unbiased estimate of model skill.



(a) Total loss

(b) Sub losses

Figure 7: Training and testing losses in one experiment of the Traffic Detection Project when the split percentage is 0.7

8.2.4.2 Training Component

As for the training component, there are some options provided for users to choose or input, such as the model arch, learning rate, batch size, number of epochs etc. After the 'start training' button is clicked, our product would start to process the training datasets i.e. generate frames from videos and created labels for each generated frame, split the dataset if users want to and then train the model based on the requirement of users.

8.2.4.3 Tracking Component

The tracking component is a relative sophisticated part. It contains two main scenarios, the first scenario is that users have split the datasets in the first two steps. In this case, there are two options provided for users to select. The first option is to track on the test datasets to check the evaluation and make model selection. The other one is that users could track their raw videos using the model they just trained and visualize them on the Cord platform. One essential thing to notice is that the two options are designed for different purposes, the first one is mainly focus on the model selection, users could decide whether using the model or not based on the statistical tracking performance. The other one places more emphasis on the intact videos visualization. Therefore, there is a trade-off between the two options, the first option has no visualization function and the second one cannot provide the statistical tracking evaluation. The second scenario is that the dataset have not been split before, then there is only one option which is tracking the intact videos and visualize them. Since the evaluation on the training dataset would result in a biased score, statistical tracking performance option would not be offered.

To sum up, the train-track procedure offers users a whole pipeline to process their datasets, train new models on their training datasets, track, evaluate their models and visualize tracked intact videos.

8.3 CLI Version

The traditional CLI version is made up of three main parts which represent three .PY files respectively: Entry point, train and mctrack(i.e. multi-class track). The flow chart of Figure 8 shows the whole pipeline of the CLI version, the whole flow is pretty similar to the API version.

8.3.1 Entry Point

Entry point is always the first .PY file that would be used in the whole pipeline. There are four main functions in this file: **gen_info**, **train_track**, **track**, and **clean**.

The **gen_info** flag is the same as the generating information function above in API version. Users need to use the flags **project** and **api** to specify the particular dataset through Cord API. The dataset information, such as labelled videos and unlabelled videos would be displayed to users such that they could check and understand their dataset.

The **train_track** flag here in the CLI version is different from the train-track procedure above. In the CLI version, this flag is only used for preparing the dataset. Users still need to manually input there project ID and API key by the flags **project** and **api** to access their dataset, the flag **dataset_selection** means dataset selection, users could use flag to choose the training dataset they want. While running the whole command line, other components of the dataset, such as ground truth would also be downloaded automatically, the selected videos will be processed and related training frames and labels would be generated at the same time. Furthermore, if users intend to split their datasets into trainining and testing datasets, they could easily specify the training percentage by **split_perc** flag. The **rand_split** flag would decide whether the training sequences are chosen randomly.

The **track** flag is similar to the directly track procedure, the core idea of this flag is for users to track a raw video by inputting a video and loading an existed model with **tracking_video_selection** and **load_model**. The **track** flag could run following the **gen_info** which means this function do not need to prepare the dataset, after downloading the videos in the **gen_info**, users could track any videos they want by loading their own models or the pre-trained models that we provide. The flag

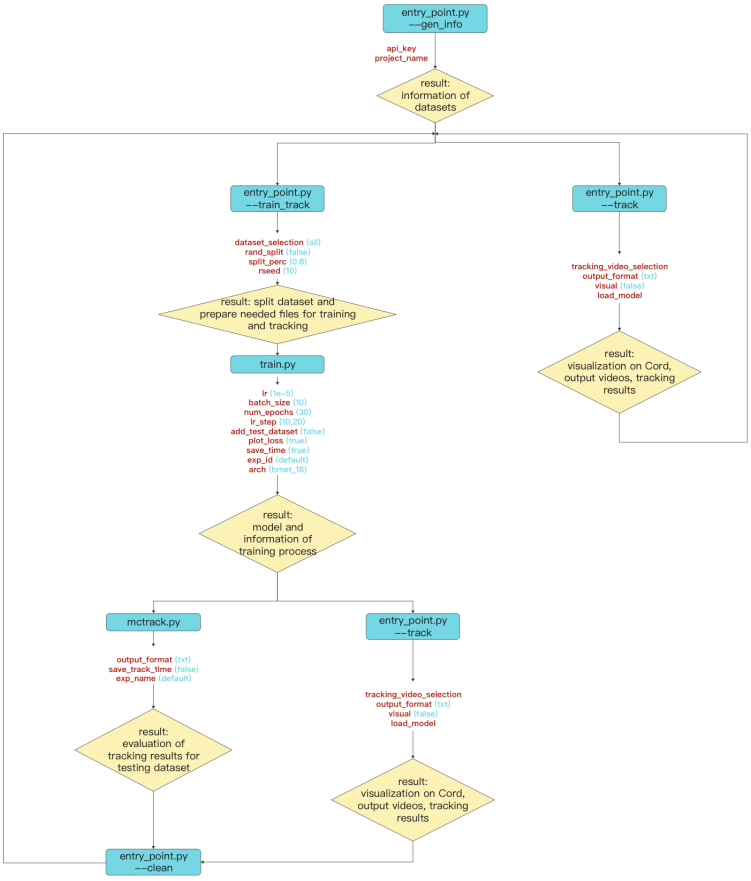


Figure 8: CLI flow chart

visual is the main visualization approach, this flag could upload the tracking results return to the Cord platform and users could view the tracked videos there.

The **clean** flag is designed for users to clean several useless files such as the files of frames paths and some redundant models.

8.3.2 Train

Training a model on the users own dataset plays an essential role in the fully pipeline. Users could control the whole training process through this file. In particular, the flag **arch** will specify the architecture of the model, there are several architectures provided such as DLA-34, HRNet-18, RESDCN-34 etc. From the experiments above, DLA-34 outperforms others for the single class and small objects tracking tasks, HRNet-18 has better performance on the multi class objects tracking tasks. The flag **lr** is the learning rate of the optimizer, **batch_size** will specify the batch size parameter for model training dataloader and **num_epochs** will specify the number of epochs to be used.

8.3.3 MCtrack

MCtrack is used for users to track their testing dataset and produce the statistical tracking evaluation in order to make the model selection. The strategy of the splitting method and the process of the evaluation have been discussed above in the API version. The program would automatically identify the testing dataset that has been splitted in the **train_track** process and evaluates the tracking results such that users could decide if the model that they trained satisfies their requirements.

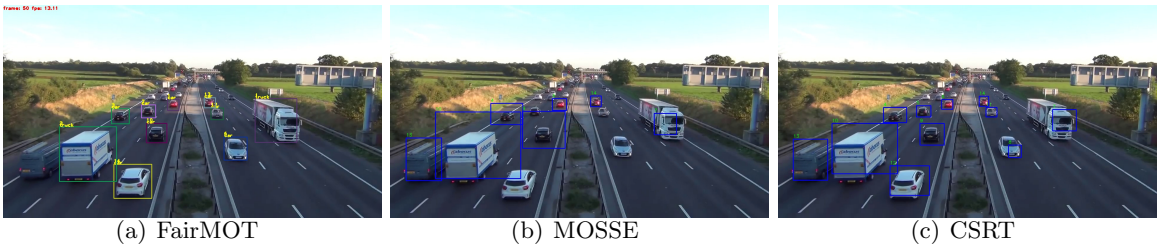
8.4 Evalutaion on Benchmarking

As a state-of-art multi-object tracking algorithm, to what degree can FairMOT outperform the traditional tracking algorithm CSRT and MOSSE? Making use of object trackers in OpenCV, we implement

benchmarking to compare the performance of CSRT and MOSSE to that of FairMOT with the same evaluation metrics. The codes are saved in the file **OpenCV_tracker.py**.

Tests are taken on the same videos as before. According to the Literature review part of these 2 algorithms, each target is required to be selected by a bounding box in its first frame. Therefore, we extract their bounding box information at the instant of their appearance in the video from the ground truth file and draw them in the corresponding frames. We also extract the object-id information and mark them at the top left corner of the bounding box. In OpenCV, one tracker can track multiple objects at the same time. However, if one object cannot be tracked in a certain frame, other objects will also be influenced, leading to the failure of whole tracker. Therefore, each object is allocated a tracker so that their tracking will process in an independent way.

During the tracking, what deserve our attention is that when the target gets out of frame, its bounding box still exists at last position it tracked or even flies to anywhere. The OpenCV documentation tells us that CSRT and MOSSE tracker are short term tracker. If we view this issue from the code level, the tracker returns a Boolean value which determines whether the object is lost or not and the coordinate of this box. When the object moves out of the frame, the algorithm wrongly updates the information and gives True value which misleads the tracker, leading to the annoying problem above. The way to mitigate this problem is to adjust the attribute psr_threshold in the function of tracker (e.g. *cv.TrackerCSRT_create( )*) and pass a higher value to it. This aims to create a tracker which is more sensitive and the tracker will not continue tracking an empty area, but will return False. Unfortunately, the attribute psr_threshold only exists in the version of C++. Finally, we add some conditions and force the bounding boxes disappear when their coordinates exceed the boundary of the frame.



(a) FairMOT (b) MOSSE (c) CSRT

Figure 9: Comparison of Tracking Results for FairMOT, MOSSE and CSRT

In this circumstance, the discussion about evaluation on benchmarking is of great significance to indicate how well FairMOT performs. After implementing the traditional method, we evaluated these tracking results via the same evaluation metrics as described above. CSRT, as a relatively accurate tracker of traditional methods, was used to compare MOTA. The table 3 shows us that the accuracy of FairMOT tracking each video is significantly higher than that of CSRT. For example, the accuracy of all videos tracked by FairMOT has reached more than 50%, while only one video of CSRT reached over 50%, with 57.76% for Light_traffic.mp4, but this video tracked by FairMOT achieved 100% MOTA. And it is obvious that overall MOTA of FairMOT is higher than CSRT, so that it has reached more than twice the accuracy of CSRT, accounting for 63.94% compared to 30.31%. The figure 9 displays the tracking result of FairMOT, MOSSE and CSRT on the 50^{th} frame of the video Highway_traffic.mp4, which is an example that can suffice to illustrate this point. When it comes to tracking time, we used the most faster traditional tracker MOSSE to make comparison with FairMOT. According to the table 4, it is clear that MOSSE takes much less time to track the object than CSRT while FairMOT is faster than MOSSE, with 79.20s and 89.32s for Highway_traffic.mp4, respectively. Therefore, we may conclude that FairMOT as a lately proposed method based on deep learning has a good performance.

We only focus on the performance of MOSSE and CSRT in this report. Additionally, to make a full comparison, the relevant python file can also implement other OpenCV built-in trackers: BOOSTING, MIL, KCF, TLD, MEDIANFLOW, to track the videos and record the accuracy and costing time. The outputs are pictures of frame (.jpg), bounding box information (.txt) and video (.mp4) of tracking results. We have included the tables of comparisons in the supplementary tables and graphs of

appendix.

	FairMOT	MOSSE	CSRT
Highway_traffic_2.mp4	0.641509434	0.06788511749	0.2417754569
Highway_traffic.mp4	0.6409478837	0.0748727565	0.3336458612
Heavy_traffic.mp4	0.5262070941	0.006071645416	0.07103825137
Light_traffic.mp4	1	-0.751552795	0.5776397516
Overall	0.6394340994	0.06094880729	0.3031358885

Table 3: Comparison of MOTA

	FairMOT	MOSSE	CSRT
Highway_traffic_2.mp4 (10 seconds)	15.80	20.97	122.87
Highway_traffic.mp4 (59 seconds)	79.20	89.32	631.18
Heavy_traffic.mp4 (16 seconds)	22.10	28.48	138.85
Light_traffic.mp4 (22 seconds)	8.70	16.87	19.92

Table 4: Comparison of Tracking Time

8.5 Model Selection

The evaluation metric is applied to conduct the benchmarking as well as the best model selection. The evaluation metrics for best models for the available dataset on Cord platform are displayed below1011. Unfortunately, since some class does not exist in certain videos for Traffic project, their MOTP scores will become None type and this will cause the overall MOTP value becomes empty. This won't be a problem for a complete dataset like the Drone project.

	IDF1	IDP	IDR	Recall	Precision	GT	MT	PT	ML
Heavy-car	0.54080	0.67073	0.45304	0.58978	0.8731707	16	2	13	1
Heavy-pedestrian	0.66987	0.8	0.57615	0.70529	0.9793103	7	3	3	1
Heavy-motorbike	0.60655	0.60655	0.60655	0.60655	0.6065573	3	2	0	1
Heavy-bus	0.91489	0.86	0.97727	0.98863	0.87	1	1	0	0
Highway-car	0.79125	0.87926	0.71925	0.72960	0.8919081	162	59	92	11
Highway-truck	0.79245	0.84782	0.74386	0.74386	0.8478260	7	3	3	1
Highway-motorbike	1	1	1	1	1	1	1	0	0
Highway2-car	0.75962	0.78759	0.73356	0.78730	0.8453038	36	21	15	0
Light-car	1	1	1	1	1	1	1	0	0
OVERALL	0.77881	0.86066	0.71117	0.73371	0.8879339	234	93	126	15
	FP	FN	IDSW	FM	MOTA	MOTP	TDt	TDa	TDm
Heavy-car	52	249	3	24	0.4991762	0.2664	5	1	3
Heavy-pedestrian	9	178	6	27	0.6804635	0.1976	3	4	1
Heavy-motorbike	24	24	0	1	0.2131147	0.2999	0	0	0
Heavy-bus	13	1	2	1	0.8181818	0.1852	0	1	0
Highway-car	1257	3844	16	305	0.6400534	0.2846	15	2	2
Highway-truck	49	94	0	8	0.6103542	0.2469	0	0	0
Highway-motorbike	0	0	0	0	1	0.3346	0	0	0
Highway2-car	252	372	3	28	0.6415094	0.2812	6	2	5
Light-car	0	0	0	0	1	0.1235	0	0	0
OVERALL	1656	4762	30	394	0.6394340		29	10	11

Figure 10: The evaluation metric for the best model for the Traffic project (some headers are shortened and some classes with no GT are deleted for the clarity of the figure)

As it can be seen in the figure, the best models we've chosen have the best scores on almost all the aspects. Starting from the Traffic project, first of all, for our first priority evaluation scores, the

best model has the highest MOTA and IDF1 among all the models. This model has a MOTA of 0.639 and an IDF1 of 0.779 while the standard scores of other models are 0.45 and 0.63, which is much less than the best model. As for the second priority, the rate of successful detection, our best model reach a score of 0.936 while the mean value for other models on this score is 0.82. This is exactly why this model is chosen as the best model.

Apart from the first and second priorities of the evaluation metric, there are lots of extra reasons for choosing this model as the best model. Considering that the car type object has the hugest number in the dataset, its tracking performance is considered as the first priority among all the class. The successful detection for the car class of this model is 0.944 which is the highest among all the models. What's more, for the classes that only have one object such as Highway-motorbike and Light-car, this best model is able to track this single object with neglectable false and miss detection. This is impossible for other models.

Apart from the advantages, this best model also has some disadvantages compared to other models. For example, although the overall score is better than others, the best model have a higher number of FN. Also, this model has a higher fragment number compared to other models which means that its detection will be less stable compared to other models.

	IDF1	IDP	IDR	Recall	Precision	GT	MT	PT	ML
Video_of_cattle_1.mp4-cattle	0.99216 34481	0.99550 68801	0.98884 23989	0.98884 23989	0.995506 8801	8	8	0	0
OVERALL	0.99216 34481	0.99550 68801	0.98884 23989	0.98884 23989	0.995506 8801	8	8	0	0
	FP	FN	IDSW	FM	MOTA	MOTP	TDt	TDa	TDm
Video_of_cattle_1.mp4-cattle	16	40	0	6	0.984379 3584	0.1615 22054	0	0	0
OVERALL	16	40	0	6	0.984379 3584	0.1615 22054	0	0	0

Figure 11: The evaluation metric for the best model for Drone project

The evaluation procedure for selecting the best model for the Drone project is the same as the Traffic project. The Drone project has a smaller dataset with fewer classes and thus our best model for this dataset has a significant-good performance. It has a MOTA of 0.984 and IDF1 of 0.990 which are close to 1, the maximum possible value for these two scores. Also, surprisingly it has a successful detection rate of 100% which is the maximum possible value. For the other evaluation scores, all of them except the MOTP are higher than the other models which indicate that this model has the best performance on almost all the aspects.

In summary, the most suitable way of selecting the best model is following the evaluation procedure described above. The first priority for the model selection are the MOTA and IDF1 scores and the second priority is the successful detection rate. After evaluating these scores, all other scores can be a supplement material for the judgement. If a certain type of functionality of the model is required, this procedure can be adjusted.

9 Conclusion

In conclusion, based on the FairMOT algorithm, we did sufficient research, divided the task groups and accomplished the software engineering project in accordance with software development strategy. We studied object detection, tracking and work classification through this deep learning method. From the establishment and containerisaton of the entire pipeline to run through an API, we have done a lot of experiments, solved various problems and found the relatively best models. After evaluating on the tracking results of both FairMOT and traditional tracking algorithms MOSSE and CSRT, we can conclude that as a state-of-the-art multi-object tracking algorithm, FairMOT presents a good performance. However, there are still some limitations of the final product. For example, when the user starts training or tracking, the web-based application has no response while the program is running in the container. It is necessary to add a progress bar in the future work so that the user could know

clearly how long the training or tracking step would finish.

References

[1] Stiefelhagen Rainer Bernardin, Keni. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008.

[2] D. Bolme, J. Beveridge, B. Draper, and Y. Lui. Visual object tracking using adaptive correlation filters. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, 2010.

[3] CaptainEven. Mcmot: One-shot multi-class multi-object tracking, 2020. `https://github.com/CaptainEven/MCMOT`.

[4] Ivan Culjak, David Abram, Tomislav Pribanic, Hrvoje Dzapo, and Mario Cifrek. A brief introduction to opencv. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 1725–1730, 2012.

[5] Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[6] Patrick Dendorfer Philip Torr Andreas Geiger Laura Leal-Taixe Jonathon Luiten, Aljosa Osep and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. *arXiv e-prints*, September 2020.

[7] Buyu Li, Yu Liu, and Xiaogang Wang. Gradient harmonized single-stage detector, 2018.

[8] Liu D. Srivastava G. et al Liu, S. Overview and methods of correlation filter algorithms in object tracking. 2020.

[9] Alan Lukezic, Tomás Vojír, Luka Cehovin, Jiri Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. *CoRR*, abs/1611.08461, 2016.

[10] Tzu-Wei Mi and Mau-Tsuen Yang. Comparison of tracking techniques on 360-degree videos. *Applied Sciences*, 9(16), 2019.

[11] ProgrammerSought. Classic mot16 data set and evaluation index in multi-target tracking.

[12] Pray Somaldo and Dina Chahyati. Comparison of fairmot-vgg16 and mcmot implementation for multi-object tracking and gender detection on mall cctv. *Jurnal Ilmu Komputer dan Informasi*, 14(1):49–64, 2021.

[13] Jack Valmadre. py-motmetrics, 2020. `https://github.com/cheind/py-motmetrics`.

[14] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. Mots: Multi-object tracking and segmentation, 2019.

[15] Chen Wang, Le Zhang, Lihua Xie, and Junsong Yuan. Kernel cross-correlator. *CoRR*, abs/1709.05936, 2017.

[16] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition, 2020.

[17] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking, 2020.

[18] Makris Dimitrios Yin Fei and Velastin Sergio. Performance evaluation of object tracking algorithms. *arXiv e-prints*, 25, October 2007.

[19] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. Fairmot, 2020. `https://github.com/ifzhang/FairMOT`.

[20] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking. *arXiv e-prints*, page arXiv:2004.01888, April 2020.

[21] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points, 2019.

A Logbook

18^{th} January, 2021: Coding starts

29^{th} January, 2021: Parser
*Video downloader, finished by Yuanping Qin.
*Extract and save video frames, finished by Jingyi Fan.
*Create files containing image paths, finished by Zhaojiang Liu.
*Create json files for clarifying the training and the testing dataset, finished by Yipengjing Sun.
*Extract necessary ground truth information from the software development kit (SDK) of Cord, finished by Ke Ding.
*Adapt the structure of ground truth files to FairMOT algorithm, finished by Yu Hong.

2^{nd} February, 2021:
*Report one, finished by Jingyi Fan, Yuanping Qin.

10^{th} February, 2021:
*Establish a virtual conda environment on Google Colab platform, configure all necessary packages, finished by Jingyi Fan, Yuanping Qin.

16^{th} February, 2021:
*Pad missing attributes in ground truth files to generate suitable data structure, finished by Ke Ding and Yu Hong.

20^{th} February, 2021:
*Do training and tracking with some pre-trained models provided by the author of FairMOT, finished by Yipengjing Sun and Zhaojiang Liu.

26^{th} February, 2021:
*Debug FairMOT built-in methods, finished by all members.

5^{th} March, 2021: Report two, finished by all members.

28^{th} March, 2021:
*Learn related knowledge about Docker, finished by Jingyi Fan and Yuanping Qin.
*Custom evaluation metric creation, finished by Yu Hong.
*Debug evaluation methods, finished by Ke Ding.
*Train and test resdcn_34, resdcn_50 based models , finished by Zhaojiang Liu.
*Train and test resfpndcn_34, hrnet_18 based models , finished by Yipengjing Sun.

31^{st} March, 2021:
*CSRT tracking algorithm implementation, finished by Jingyi Fan.
*MOSSE tracking algorithm implementation, finished by Yuanping Qin.
*Adapt new evaluation metrics to MOSSE and CSRT methods, finished by Ke Ding.
*Debug new metric based on light traffic datase; check gt files and model outputs, finished by Yu Hong.
*Do evaluation on different models, finished by Yipengjing Sun.
*Generate output videos, finished by Zhaojiang Liu.

4^{th} April, 2021:
*Solve issues of failure detection, remove the bounding box of out-of-frame object, finished by Jingyi FAN.

*Evaluate the performance of CSRT and MOSSE, finished by Yuanping Qin.
*Optimise the training method, finished by Yipengjing Sun and Zhaojiang.
*Built the MCEvaluator class to enable true multi-class evaluation, finished by Ke Ding.
*Modify Pipeline into modules, finished by Yu Hong.

7^{th} April, 2021:
*Set up the automated pipeline, finished by Ke Ding.
*Relate pipeline to visualisation platform provided by Cord.tech, finished by Yu Hong.
*Develop train-split strategy, finished by Yipengjing Sun.
*Plot training and testing performance (e.g. average loss per episode) against portion of split, finished by Zhaojiang Liu.

18^{th} April, 2021:
*Dockerfile compiling, finished by Yuanping Qin.
*Build docker image and debug Dockerfile, finished by Jingyi Fan.
*Set up the environment of Google Cloud platform, finished by Zhaojiang Liu.
*Apply the dockerisation of pipeline on Google Cloud platform, finished by Yipengjing Sun.
*API application design, finished by Yu Hong and Ke Ding.

22^{nd} April, 2021:
*Code the API functions, finished by Yu Hong.
*HTML structure design and coding, finished by Ke Ding.
*Adapt API functions to the pipeline, finished by Yipengjing Sun.
*Transform data structure to fit the API, finished by Zhaojiang Liu.
*Literature review of FairMOT, finished by Yuanping Qin.
*Literature review of MOSSE and CSRT, finished by Jingyi Fan.

B Minutes of Group Meetings

1^{st} meeting, 10:00 a.m. GMT, 20^{th} December 2021:
*Introduction of the project, group members, supervisor and industrial partners.
*Do some reading.

2^{nd} meeting, 13:30 a.m. GMT, 4^{th} January 2021:
*Organise in groups of 2 or 3, with each group having a specific area of focus.
*Look through the project, split up all the work into two weeks sessions, specify the responsibility of each member.
*Work out a plan, send it to industrial partners for review, discuss it in the next meeting.

3^{rd} meeting, 8:00 a.m. GMT, 18^{th} January 2021:
*One people of the backend team can work on dockerization, and the other two people can work on the parsing.
*Research group can work on getting some kind of metric script and some OpenCV trackers working as a benchmark.

4^{th} meeting, 4:00 p.m. GMT, 1^{st} February 2021:
*Start to set up the pipeline.
*Do some modifications to Report 1.

5^{th} meeting, 4:00 p.m. GMT, 16^{th} February 2021:

6^{th} meeting, 3:00 p.m. GMT, 2^{nd} March 2021:

*Establish own evaluation metrics, instead of relying on the metrics provided by FairMOT.
*Double check the ground truth file.

7^{th} meeting, 12:00 p.m. GMT, 29^{th} March 2021:
*Continue to debug the evaluation functions.
*For benchmarking, there is no need to classify the objects in the video. We only need to use OpenCV to do tracking.

8^{th} meeting, 12:00 p.m. GMT, 5^{th} April 2021:
*Split videos and only a part of them for training.

9^{th} meeting, 12:00 p.m. GMT, 12^{th} April 2021:
*Create a virtual machine on Google cloud to test the pipeline with the use of GPU.

10^{th} meeting, 12:00 p.m. GMT, 19^{th} April 2021:
*Add exception classes in the codes.
*Use Python Flask to construct a web based API application. Make it user-friendly.
*Send back numerical results, plots (e.g. frames) are not necessary. But plots can be shown on the web.
*Some parameters like learning rate or batch size can be option. Put them in advanced settings and give each of them a default value.

C Detailed Work Breakdown

More details can be seen in the previous 'Division of work' and 'Logbook' section.
Flexibility shall be reserved, groupings subject to change as project progresses.

D Supplementary Tables and Graphs

	FairMOT	MOSSE	CSRT	TLD	Medianflow	Boosting	MIL	KCF
Highway_traffic_2.mp4	0.641509434	0.06788511749	0.2417754569	-0.3942558747	0.6626631854	-0.4135770235	-0.4120104439	0.2924281984
Highway_traffic.mp4	0.6409478837	0.0748727565	0.3336458612	-0.529533887	0.02149745513	-0.3631797482	-0.3982721672	-0.2141039379
Heavy_traffic.mp4	0.5262070941	0.006071645416	0.07103825137	-0.2908318154	0.2137219186	-0.7413479053	-0.8275652702	0.1706132362
Light_traffic.mp4	1	-0.751552795	0.5776397516	0.6273291925	-0.1428571429	-0.5900621118	-0.5403726708	-0.5403726708
Overall	0.6394340994	0.06094880729	0.3031358885	-0.4845885821	0.1028678638	-0.4036987403	-0.4388099705	-0.130956848

Table 5: Comparison of MOTA between FairMOT and Traditional Trackers

	FairMOT	MOSSE	CSRT	TLD	Medianflow	Boosting	MIL	KCF
Highway_traffic_2.mp4	15.80	20.97	122.87	764.15	54.35	157.14	299.46	24.73
Highway_traffic.mp4	79.20	89.32	631.18	3264.49	185.50	981.28	1309.21	167.52
Heavy_traffic.mp4	22.10	28.48	138.85	608.76	59.69	204.54	223.82	39.12
Light_traffic.mp4	8.70	16.89	19.92	43.50	23.64	21.60	26.24	31.07

Table 6: Comparison of Tracking Time between FairMOT and Traditional Trackers