Computer Vision Final Project: Object Tracking in Python

Rosalie Didcock
December 15, 2021

1 Introduction

In this experiment project, I worked with Bryan Uribe to implement and compare two different methods of object localization on the same video dataset. Overall, our goal was to accurately predict the true position of objects in motion. We focused on a specific dataset which captures diverse and real-life sequences. In these videos, our models combined image processing and two distinct methods of object tracking to predict the target object's position at each individual video frame. After generating predictive sets containing the bounding boxes for the objects in motion from the video frames using my object tracking method, I quantified the accuracy of my methods by comparing the model's predictions to the true motion, as displayed in the video frames themselves. Then, I passed our model's predictions through a Kalman filter to analyze the improvement of my model's accuracy by drawing on outside object tracking methods. My object tracking methodology was originally going to replicate an academic paper, but I ended up combining a number of methods from class and outside research and diverging from this initial plan.

Object tracking is an important field in computer vision, with potential applications in self-driving cars, security cameras, traffic control, medical imaging and video editing. These areas are of high interest and we wanted to work on models which are directly used in these fields. In particular, we wanted to evaluate multiple models and how they work in various scenarios. In this project we investigated how successfully the tools we built can extract object features over multiple frames. We also wanted to tackle prevalent issues with frame to frame object detection, and investigate different real-world scenarios, such as when the target object passes behind another object in a scene or blurring or warping of the target object from the motion of the object and camera. In short, this project allowed us to get a sense of how different

techniques can be used to overcome object-tracking obstacles.

2 Project Implementation

2.1 Project Overview

My object tracking method makes use of a number of techniques, including frame differencing, morphological convolutions, masking, Mask R-CNN, and kalman filtering. The training data is passed into the Mask R-CNN model as the training set with the given annotations as ground truth bounding boxes. The testing data is passed into the model using the output of the frame differencing method as the ground truth bounding boxes, where the given first frame annotation is the first bounding box in the result.

I first compared the measurements to the true positions from the data, then applied a Kalman filter using my bounding box data as measurements to further attempt to accurately track the true position of the object. By implementing a Kalman filter and inputting position data as measurements, I was able to use a simple constant velocity model and update it at each step with the measurement models outputted by our frame to frame object localization methods to achieve smoother object tracking.

2.2 Dataset: TrackingNet

We used TrackingNet, a publicly available dataset which holds videos and bounding box annotations for 27 classes of objects. This dataset includes realistic and diverse videos, as the sequences are pulled from YouTube. Moreover, this dataset holds 30,132 training sequences and 511 testing sequences with annotations, which will be used for the training and testing of our models. For the training data, bounding boxes are given for each frame of video, and for the test data bounding boxes are only given on first frame.

The table below contains information on how the TrackingNet dataset compares to other datasets often used in object tracking.

Table 1. Comparison of current datasets for object tracking.

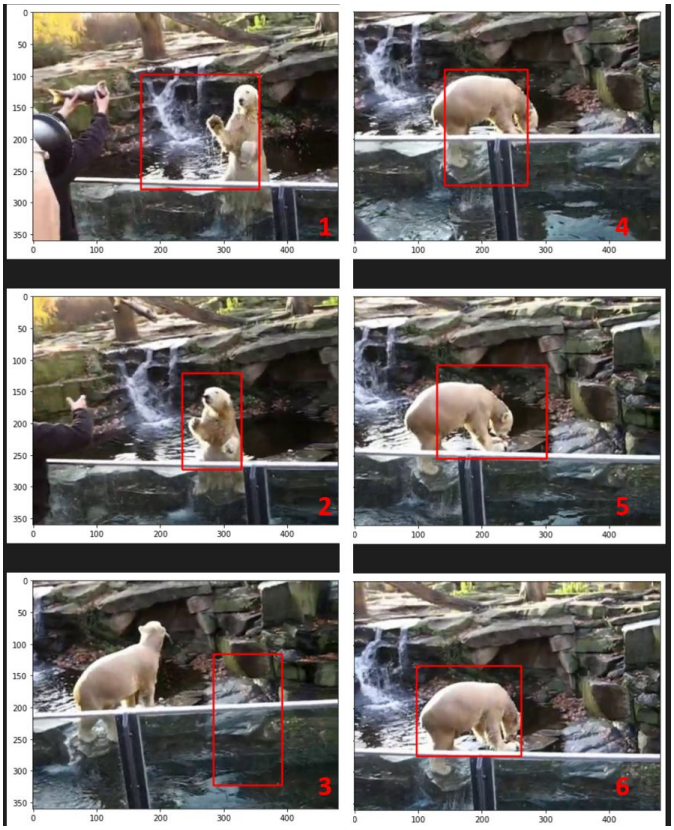
Datasets	Nb Videos	Nb Annot.	Frame per Video	Nb Classes
VIVID [5]	9	16274	1808.2	-
TC128 [33]	129	55652	431.4	-
OTB50 [48]	51	29491	578.3	-
OTB100 [49]	98	58610	598.1	-
VOT16 [22]	60	21455	357.6	-
VOT17 [23]	60	21356	355.9	-
UAV20L [36]	20	58670	2933.5	-
UAV123 [36]	91	113476	1247.0	-
NUS PRO [29]	365	135305	370.7	-
ALOV300 [43]	314	151657	483.0	-
NfS [13]	100	383000	3830.0	-
MOT16 [35]	7	182326	845.6	-
MOT17 [35]	21	564228	845.6	-
TrackingNet (Train)	**30132**	**14205677**	**471.4**	**27**
TrackingNet (Test)	**511**	**225589**	**441.5**	**27**

In the TrackingNet dataset, there are 12 folders of training data, and 1 folder of testing data. The following screen clipping shows the structuring of the dataset.

```
TrackingNet
 - Test / Train_X (with X from 0 to 11)
   - zips
   - frames
   - anno (Test: annotation only for 1st frame)
```

As I did not get access to the GPU for testing, I trained on very small subset of the data (that my computer could handle). I pulled 35 videos from the `Train_0` folder, and 15 videos from the `Test` folder.

While working on this project, I encountered some strange behavior in the annotated training data, which I believe resulted in complications in model training (discussed further in Section 3). While some of the bounding boxes appear to accurately capture the object of interest, many seem to capture almost exclusively background information. Over reviewing numerous training sets, this behavior is consistent across the set. The figure below illustrates this behavior. Using a training video from the TrackingNet dataset, these images show every 60^{th} frame for the first 300 frames in the video and the given bounding box annotations.



Note: Bryan did not have this issue, and I am pretty sure something just went wrong when I was trying to download it, since I struggled more than Bryan did. Unfortunately I did not have

time to verify this.

2.3 Frame Differencing Implementation

The frame differencing method I implemented takes a number of steps, making use of a few techniques from class, but inspired by the original paper and methodology I planned to follow (Weng, Shiuh-Ku Kuo, Chung-Ming Tu, Shu-Kang, 2006, *Video object tracking using adaptive Kalman filter*).

1. I used frame differencing on 3 consecutive frames from the second frame in the dataset and on. I used the combined difference between the current and previous frames, and current and next frames to extract the 2D image features of the moving objects.

2. I then applied a threshold to the image, turning "off" all pixels with less than a certain frame difference value.

3. Using a kernel of size 9, I applied a morphological closing to the image, or a dilation followed by an erosion to remove noise from the background of the image.

4. Using the bounding box annotation from the previous frame, we can choose to focus on a region of interest to improve our guessing and ignore unwanted movement in the video. I chose to turn "off" the pixel values for all pixels further than 20 pixels outside of the previous frame's bounding box.

5. Finally, I find the contours, specifically the largest contour in the resulting image, and generate a final predicted bounding box for the current frame that encapsulates it.

2.4 Mask R-CNN Model Training and Implementation

For my Mask R-CNN model, I used an open source repository as the starting point, and adapted the structure of the model to account for the TrackingNet data, as well as to be able to load in the output of my frame differencing method. The repository is available at `https://github.com/leekunhee/Mask_RCNN.git`. I directly cloned the repository into the working repository for my final project so that I could make these adjustments as needed.

There are two different modes for the Mask R-CNN model that we use, training mode and inference mode. For model training, we load in our dataset object using the built in MASK R-CNN class definition. Mask R-CNN uses bounding box and class regression to generate bounding boxes, a branch for predicting an object mask (Region of Interest) in parallel with the branch for bounding box recognition. After pre-training my model with MS COCO weights, I used the TrackingNet dataset to train my model for 3 epochs, with 131 training steps per epoch. This generated a configuration file that could be loaded into the inference model as weights for model detection. The TrackingNet dataset I used did not contain dataset class annotations, thus only 2 classes were present in the model: object and background.

2.5 KALMAN FILTER IMPLEMENTATION

Kalman filters have many applications, and in computer vision can be used for object tracking to predict an object's future location, and reduce the noise introduced by inaccurate measurements.The Kalman filter works in two steps, the prediction state and the update step. The prediction step uses a predefined process model, in this case using a simple constant velocity model, and the position and uncertainty information from the previous time step. In the update step, measurement information is taken into account and the predicted position and covariance are "updated" with the observations passed into the model. Kalman filters operate on systems in linear state space format, where

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$
$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

The variables are the following:

- $\mathbf{x}$: the state vector, for this problem containing x and y position.
- $\mathbf{y}$: the output vector, contains what you are able to measure.
- $\mathbf{u}$: the input vector, containing information on system dynamics that are not estimated as states, velocity vector in this problem.
- $\mathbf{w}$: the process noise vector, models uncertainty in actual equations used in estimation model, covariance Q.
- $\mathbf{v}$: the measurement noise vector, models noise in sensor measurements, etc, covariance R.
- $\mathbf{F}$: matrix containing coefficients of state terms in state dynamics.
- $\mathbf{G}$: matrix containing coefficients of input terms in state dynamics.
- $\mathbf{H}$: matrix that converts states to outputs, reduces state to what is measureable.
- $\mathbf{Q}$: covariance of the process noise vector
- $\mathbf{R}$: covariance of the measurement noise vector

The two steps are defined by the following equations:

Predictive Step:

$$\mathbf{x}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} \quad \text{state vector initial estimate}$$
$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^{T} + \mathbf{Q}_{k-1} \quad \text{state vector initial covariance estimate}$$

Corrector Step:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \quad \text{Kalman gain}$$
$$\mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k|k-1}) \quad \text{corrected state}$$
$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \quad \text{corrected state covariance}$$

I implemented a Kalman filter in python to take in the bounding boxes output by our object detection and tracking methods as measurements.

3 Method Evaluation

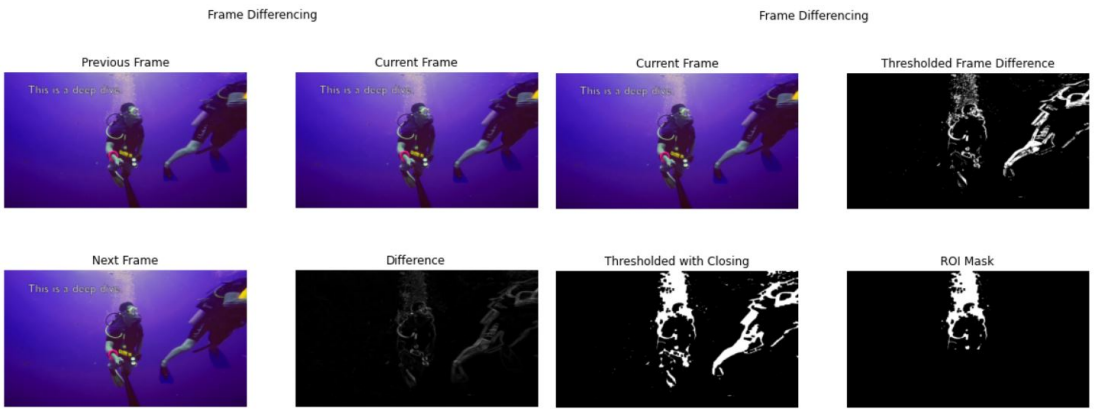
Due to some complications in the Mask R-CNN training (as a result of lack of video data as well as issues with incorrect ground truth annotations), I wanted to evaluate the results from the frame differencing contour generation method for the testing data as well.

3.1 Frame Difference Method Results

In order to evaluate my method, I outputted figures every 60 frames of the image itself with the predicted bounding box overlaid. I also plotted the final frame difference image with the predicted bounding boxes, illustrating what contour the bounding box was drawn around. In order to further understand how this method was working, I also looked at the frame difference image at the intermediate steps I took, outlined in section 2.3. The following figure demonstrates the generated bounding boxes for 6 frames of a video in our test data set (where only the first frame annotation is provided) and the corresponding plots of contours that were found to generate those bounding boxes.

Taking a look at how the first figure was generated, the next two plots go more in depth on the frame difference bounding box generation, and how the different steps I perform affect the contour that is designated as the object in motion. The 4 subplots on the left show the 3 consecutive frames used, with imperceptible differences, and the corresponding combined frame difference. The 4 plots on the right illustrate how thresholding, closing, and masking the region of interest using the annotation from the previous frame help to improve the contour and bring it closer to the true object in motion shown in the frame.

In terms of accuracy, on this testing video, my method performs very well, and results across the frames for that video matched up well with the bird in the actual video frame. Even with extremely subtle differences between frames, my method pulled out those differences, and the transformations that I made managed to create contours that almost fully captured the silhouette of the bird. This video was especially easy for my method to predict due to the limited background noise in the frames, and other than the appearance of a tree in a few, it was a relatively easy task to detect a dark object in motion against a monochromatic and muted background. I saw less accurate predictions on other testing videos in the dataset, but overall my method was fairly accurate in its estimations.

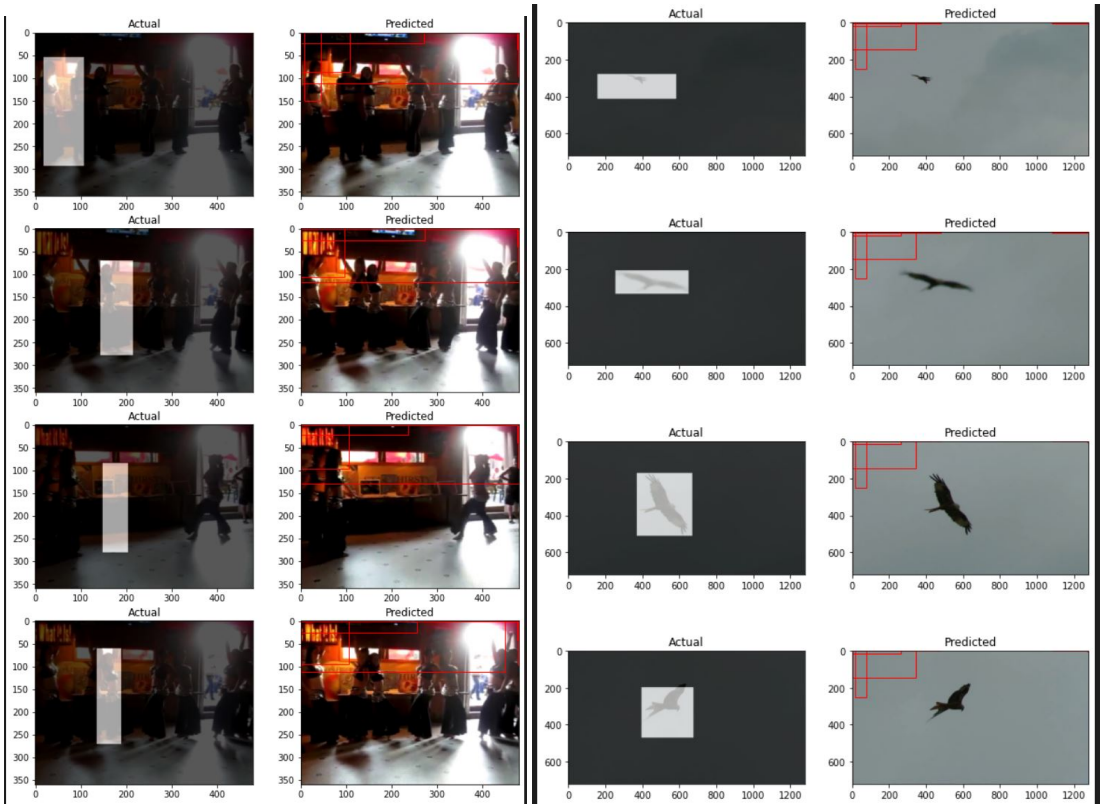


This is another example from the testing dataset of my frame differencing method, demonstrating the same outputs as the previous figure. There is a second diver that is also identified

with the raw combined frame difference, as well as interference from the light and bubbles around the diver. The operations my method performs manages to filter out the second diver, focusing back in on the object of interest. However, the light above the divers head is included in the final contour, skewing the predicted bounding box position upwards.

3.2 Model Training and Results

I managed to adapt the MASK R-CNN model so that it successfully took in input and outputted bounding boxes, but unfortunately the training that I was able to perform was not extensive enough or exhibited some large flaws. Since the training data contained a lot of background information (as mentioned above), the model was trained poorly in how to classify object from background in motion, and the regression resulted in bounding boxes centering on areas in the background. I was also unable to run a very extensive or diverse set of data, making it difficult for the model to train its weights effectively on a range of data and causing overfitting for the object in motion it did train on. The two images below display "Actual" and "Predicted" bounding boxes for a video from the train (left) and test set (right).



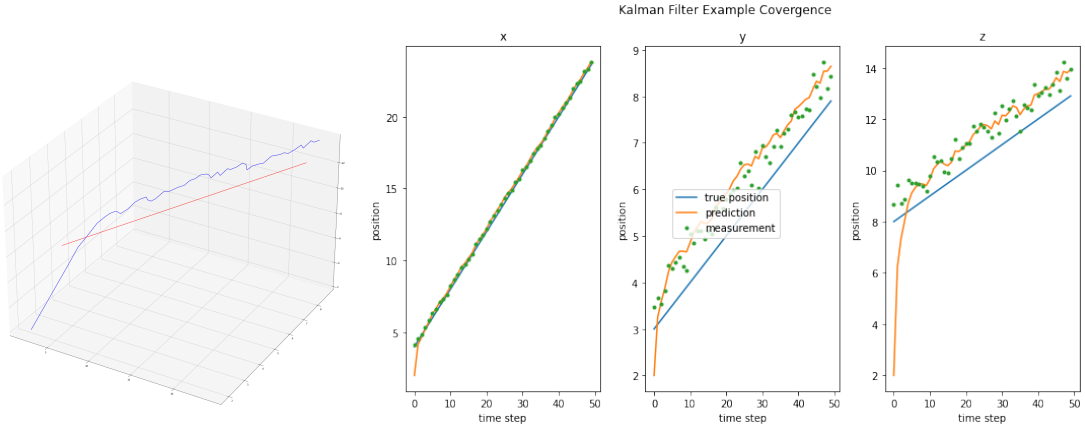
We can again see that for the training video, the "Actual" bounding boxes given by the TrackingNet dataset are inaccurate. Keep in mind that the "Actual" bounding boxes for the testing example are generated by the frame differencing method. In both of these examples, multiple bounding boxes are generated (not all pictured), and the background in the upper

left corner is where all the predicted bounding boxes are concentrated. The output for the MASK R-CNN model after training was fairly unusable, and I would like to revisit this in the future with the entirety of the dataset.

3.3 Kalman Filtering

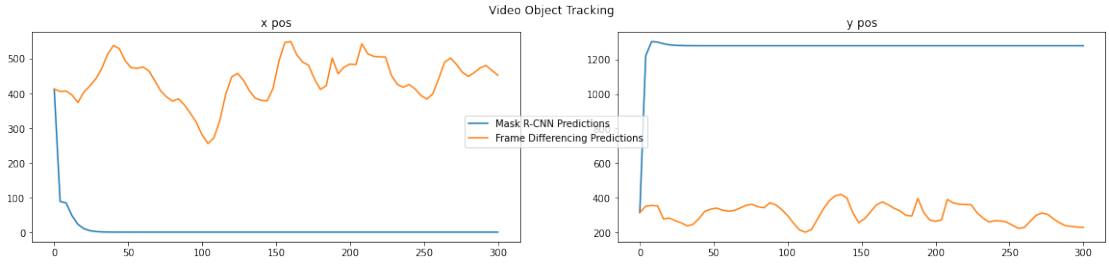
3.3.1 Filter Example

I implemented my Kalman Filter, then tested it initially on a simple example to make sure that my prediction and update steps were set up correctly. I simulated a particle moving in a straight line, then generated measurements to use in the update step from the process and noise matrices I defined. The following image displays the results of the example I ran through my filter, illustrating the convergence to the true position, skewing towards the measurements as they are used to update the prediction. The functionality of my Kalman filter is verified in the figure below. The convergence of the filter's prediction to the true position can be seen in the figure below in both the 3D plot of position over the time steps and the flattened 1D individual position graphs.

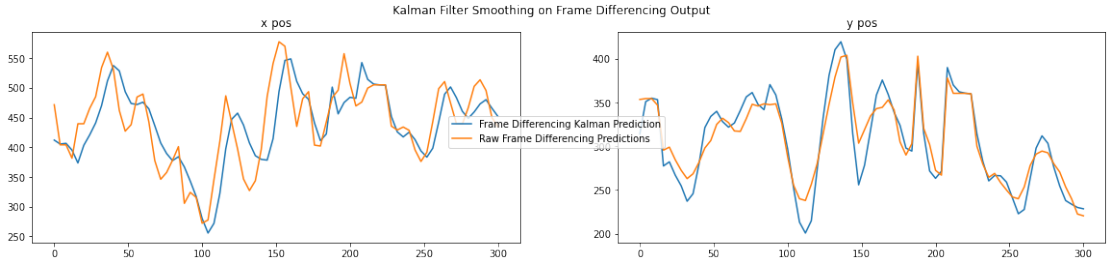


3.3.2 Passing Bounding Box Data Through Kalman Filter

To simplify the filtering process, I chose to extract x and y coordinates from the center of each bounding box measurement, and track the 2d position of the object in the video frame. The following plots display the x and y positions of the object in motion in the video over the frame number, for the predictions outputted by the Kalman filter for just the frame differencing method as well as for the Mask R-CNN detections.



Due to the bad predictions from the Mask R-CNN model, the positions converge very quickly, as there is limited variation in the measurements. Focusing on the orange line, I replotted the raw x,y position coordinates from the frame differencing method alongside the filtered path prediction.



The Kalman filter smoothed the object tracking data, although there is still noise in both directions. While some of this pattern is explained by the motion of the bird in the video, there is still error that should be quantified. The testing dataset contained no ground truth annotations, and extracting those is necessary to quantify the error here, and how much filtering improved the predictions.

4 Conclusion

While I was not able to successfully train my Mask R-CNN model due to limited time and the complications with the resulting subset of data I collected, I was able to implement frame differencing method to use for intial predictions that successfully captured object positions, as well as implement a Kalman filter with a constant velocity process model create a smoother predicted path of motion while taking model detection data into account.

4.1 Next Steps and Future Work

The next step that I would take is addressing the issues with my training data, or surpass that and train on the entire dataset with the GPU. The TrackingNet dataset contains 27 classes of objects in motion, and I also did not have access to the class data for each of the training videos, limiting my model to only classifying object from background. Obtaining that data and expanding my model to be able to classify objects in motion as different classes of objects would improve model performance and recognition, especially with the incredibly diverse range of videos emcompassed in the dataset.

Due to time constraints, I was only able to use qualitative data to evaluate the performance of my model. The next steps I would take would be to quantify my results better. For the frame differencing method, I would test it on ground truth bounding boxes and the training dataset (once the issues with it are resolved). Determining an average over the entire dataset in how accurate this method is would be very useful in determining how the parameters involved should be adjusted, such as the size of the morphology kernel and the threshold. I would also looking into the bounding box loss and class loss of my model, and optimize them by adjusting training settings.