Multi-Domain Convolutional Network for Multi-Objects Tracking and its Application on Traffic Monitoring

Abstract

We propose a visual tracking algorithm based on multi-domain convolutional network, which was designed to solve single-object tracking problem. Our algorithm need a pretrained detection network to indicate all objects in a frame as groundtruth. Structure of tracking network is quite conventional, three convolutional layers as shared layers and multi-domain three fully connected layers for each object. When an object is detected for the first time, we train shared layers of network and corresponding branch of it. If there are more than one newly detected objects, we train the network with respect to each domain iteratively. During online tracking, trainable variable are merely parameters in fully-connected layers in specific domain. The proposed algorithm is proved to have great performance in vehicle tracking problem, we apply it to a demo rude-driving detection application.

1.Introduction

Before Neural Networks are widely applied, people applied conventional machine learning to complete regression or classification task. However, performance is limited by features extracted by other analytical tools. In fact, when input object is complicated, it is almost impossible to design a prefect feature extractor.

In image processing, we know filtering is actually feature extracting procedure, for example, low pass filter extracts low frequency information from source. Filtering is performed by convolution operation between source and kernel. Thus, convolutional layers in network can be regarded as feature extractor, except for kernels are not analytical designed but optimized during training via back-propagation, while fully-connected layers, they can be seen as a mapping function between feature and output.

CNN has proved its efficiency in image classification task, although we can treat every object as an individual class and track them by simply detected them in each frame, but such network will be too large and can not handle either new object coming or similar objects issue, two identical vehicles, etc.

In general, multi-objects tracking can be viewed as a multi-variable estimation problem[1]. Given an image sequence, say S(t,i) denotes the state of i-th object in t-th frame. St = {(S(t,1)),(S(t,2))...(S(t,M))} denotes states of all M objects in the t-th

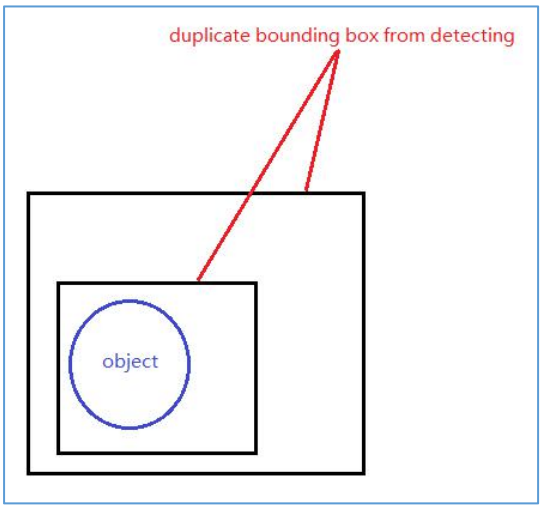
frame. Si = {S1i,S2i...SEi} denote the sequential states of i-th object, where S1 is the first frame and SE is the last frame.S[1:t]={S1,S2...St} denotes all the sequential states of all the object from the first frame to the t-th frame.

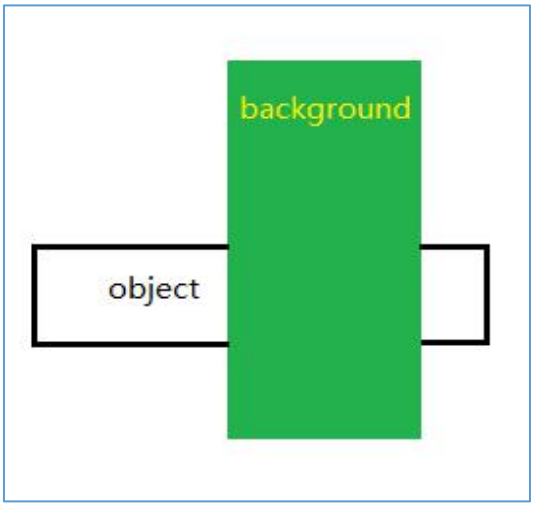
Our algorithm is DBT(detection based tracking), hence we utilize o(i,t) to denote the collected observations for the i-th object in the t-th frame, Ot={o(1， t)...o(M,t)} to denote the collected observations for all M objects in the t-th frame, and O[1:t] denotes all the collected sequential observations of all the objects from first frame to the t-th frame.

The objective of multi-object tracking is to find the optimal sequential stated of all the objects, which can be generally modeled by MAP: Sh[1:t] = argMax P(S[1:t]|O[1:t]).

Tracking source is usually in large-scale, which contains a wide combination of targets and background. Challenges of multi-objects tracking in such scale includes but not limited to difference of moving pattern, appearance, scales, speed, adding new object, delete disappearing object, occlusion and deformation. In our practice, we also find performance of online tracking is limited by performance of detection network a lot as well. In addition, compare to training samples in other task such as classification, quantity of samples for online training is much smaller, hence ordinary learning methods based on regular classification doesn't work in this case.

Previous work, multi-domain network(MDNet), is designed to track single object and proved itself on many benchmark dataset. Our network, based on MDNet, contains shared pretrained convolutional layers as common feature extractors and bunch of individual branches for each object just as MDNet does. Each branch is a three-layers fully-connected network, whose parameters are going to be updated during online tracking. However, to let network have capacity to track multiple objects still need a lot of work. For example, network should be able to discriminate whether object is correctly detected, temporarily occluded by other objects or has already left region. In addition, detection network sometimes provides wrong ground truth, a typical error is considering a large vehicle, a truck for instance, as two vehicles. Tracking network should be able to handle those kind of mistakes.





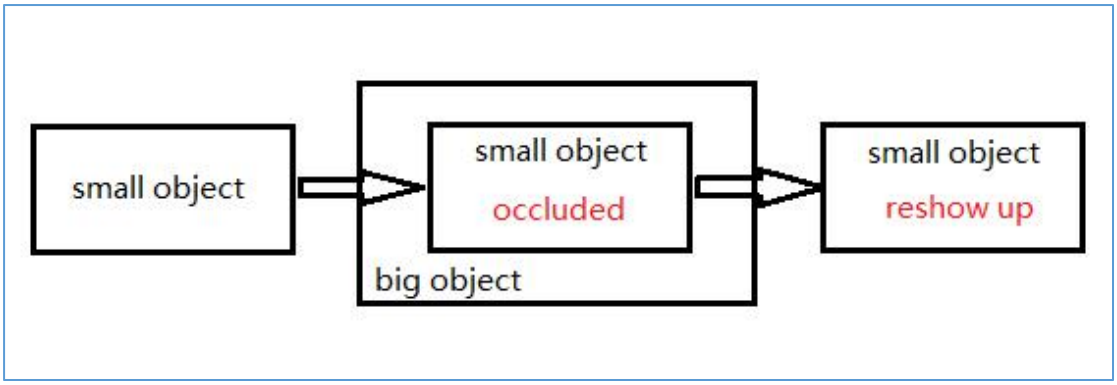


Figure 1.1 Common tracking challenge: duplicate detection from detecting network, object being occluded by background, object being occluded by other object (partly or completely)

The rest of the article will be organized as followed: In section two, there will be a brief review of related work MDNet. In section three, our own network will be described. In section four, the whole algorithm for vehicle tracking will be discussed, including detection network and how to cascade our network with it. In last conclusion section, experiment result will be posed and we will discuss a little more about other potential application.

2. Previous Related Work: MDNet

MDNet is designed to track single object in large-scale sequence. Now let's do a quick review about it on key concepts.

2.1 Framework

Its architecture is shown in Figure 2.1.1.

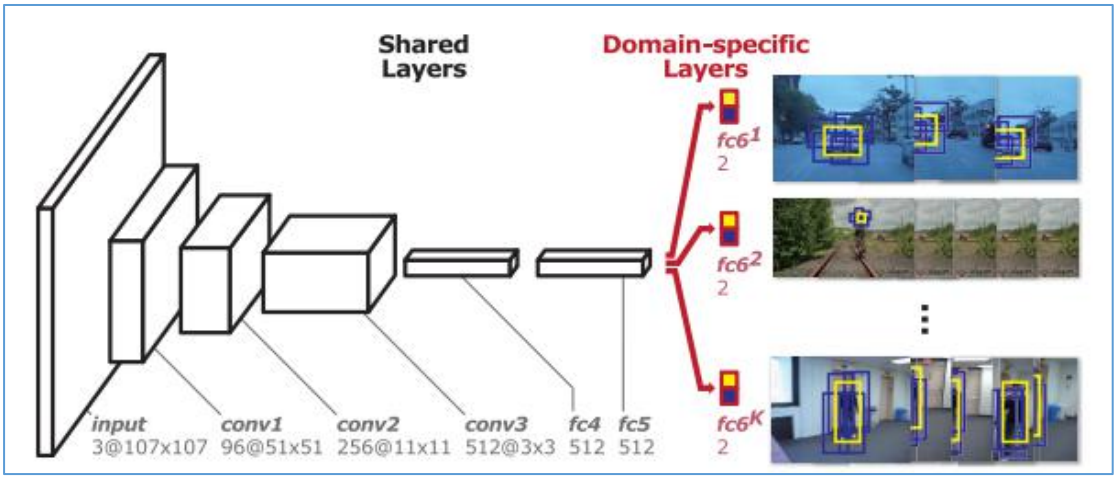


Figure 2.1.1 Architecture of MDNet (pretrain)

The size of input bounding box is 107x107. Network has five hidden layers

includes three convolutional layers and two fully-connected layers. The branches are cascaded with layer five. Each of the K branches contains a binary classification layer with softmax cross-entropy loss.

2.2 Overlapped ratio (IoU)

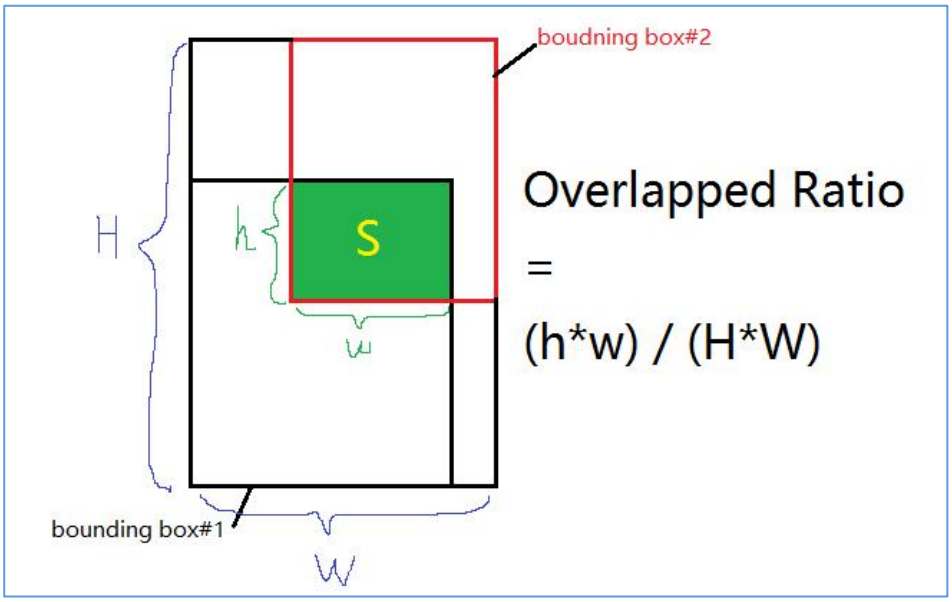


Figure 2.2.1 definition of overlapped ratio

2.3 Positive Sample and Negative Sample

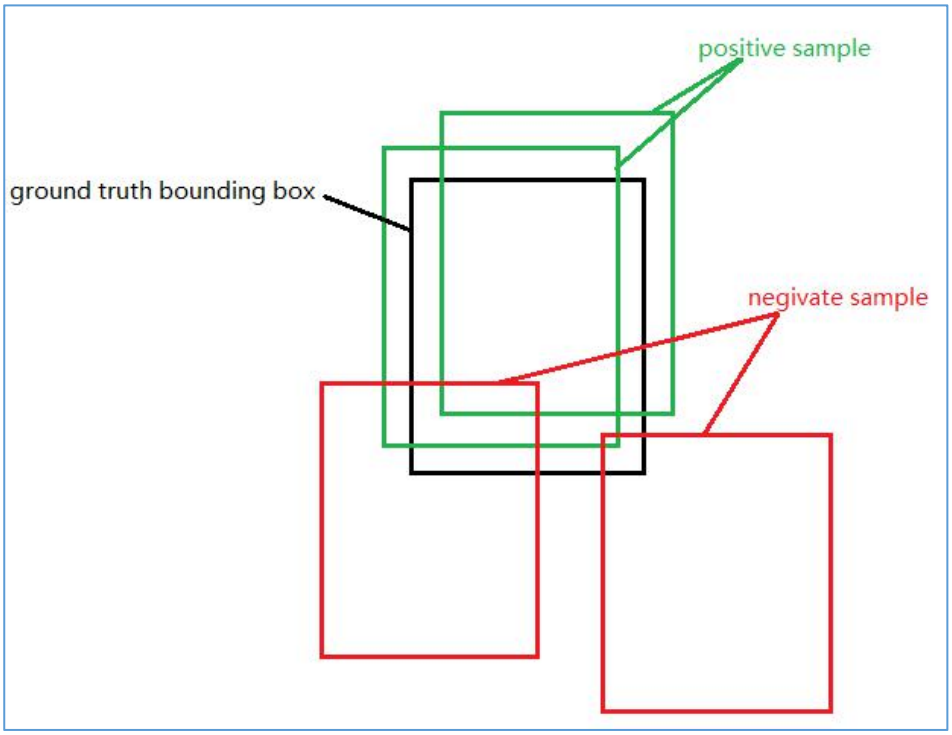


Figure 2.3.1 positive sample and negative sample

If overlapping ratio of a generated bounding box is higher than threshold, it will

be considered as positive sample with final output label (1, 0); otherwise it will be considered as negative sample with final output (0, 1).

2.4 Long-term update and short-term update

Assume L denotes period of long-term update and S-denotes short-term update. Long term update means for every L frames, whole network will be updated except for those who have already been regarded as invalid. Short-term update happens only when the tracking of an object fails (Target score is smaller than threshold). Short-term update merely update the branch of current object. Samples are collected during successful tracking. Whenever an object is successfully detected, algorithm will use it to generate bunch of samples as shown in Figure2.3.1 and store samples in pools for later usage.

2.5 Bounding Box regression

Given the first frame of a test sequence, train a simple linear regression model to predict the precise target location. In subsequent frames, adjust target locations estimated from reliable tracking. Due to time consuming issue, regression will just be applied in the first frame.

3. Our Network: MDMONet

To satisfy more complicated condition in multi-objects tracking, we propose MDMONet. Recall the challenge we mentioned in section one, network should be robust enough to handle occlusion/reshow up issue and bad detection from detecting network.

Besides, coming in/leave issue is another factor we need to consider. When an object is confirmed to have left monitor region, it is better to delete everything about it or even an 1000 frames sequence can leading to memory consuming issue.

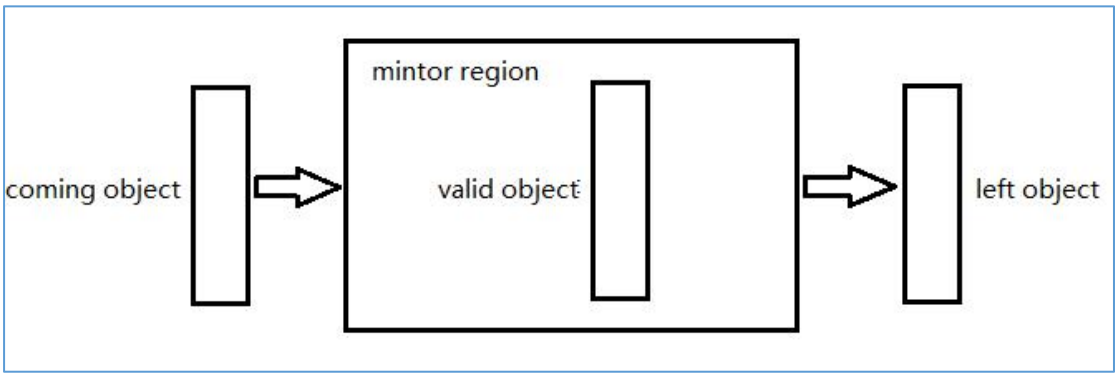


Figure 3.1 object coming /leaving

3.1 Framework of MDMONet

Most of the construction is identical as MDNet. The biggest difference is we cascade a branch of fully connected layers after convolutional layer instead of single

fully connected layer branch.

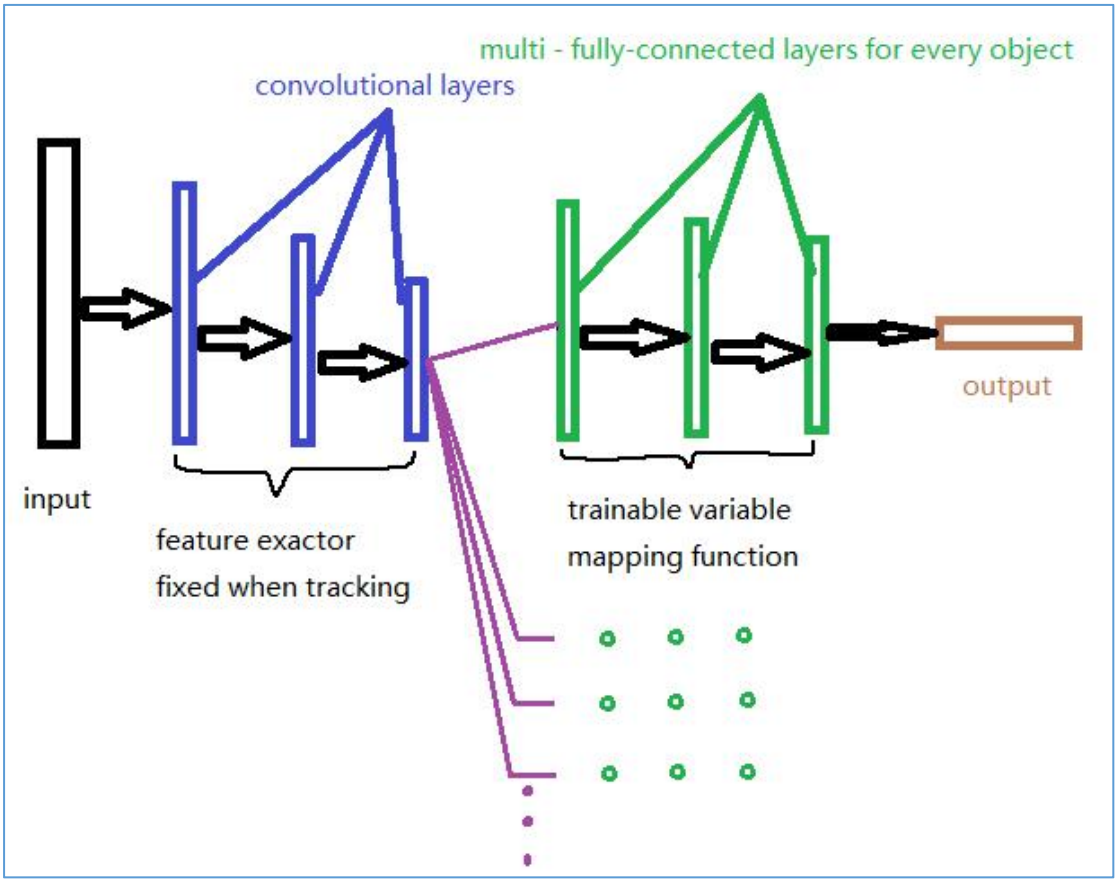


Figure3.1.1 Architecture of MDMONet

Use all 5 layers including two fully-connected layers as shared layers doesn't make sense to us. Since convolutional layers product features and fully-connected layers provides mapping, object information left after fc5 in MDNet is too less for complicated multi-objects tracking task afterward. Beside, due to single fc6 layer's complexity is too less, performance declines even more. Our experiment shows unrelible outcome if keeping MDNet's original structure. As shown in Figure3.1.1, we cascade three fully-connected layers directly after convolutional layers, thus there will be 'high quality' features available and strong enough mapping network to distinguish each object.

3.2 Deal with duplicated detection of detecting result

Before a new object is stored to the pool, we need to check if it is duplicate with existing bounding box. The judgement standard is overlapped ratio as described in Figure2.3.1. Strategy here is pretty straight forward, that is, if current bounding box has overlapped ratio higher than threshold (usually is very small) with existed bounding box, it will not be considered as an object.

3.3 Tracking method

For every objects, we have three flags as their status:'justfi', 'justfi_no' and

'justfiF'. justfi means we track current object successfully; justfi_no means the number of continuous frames we failed to track current object up to current frames for current object and justiF means current object is permanently deleted, thus do not need to be considered. In addition, we record every detected bounding box for them.

Notice that we keep each object's status and 'trace' alive, even it is confirmed leaving or lost, we only delete its sample clusters and model.

Suppose we are in the middle of tracking, network is computing one object in one frame. First thing to do is checking whether it is still valid via flag 'justiF'. If it is already out of scope, network will ignore and skip it; else it will perform tracking. Tracking is always based on the position recorded in last frame first.

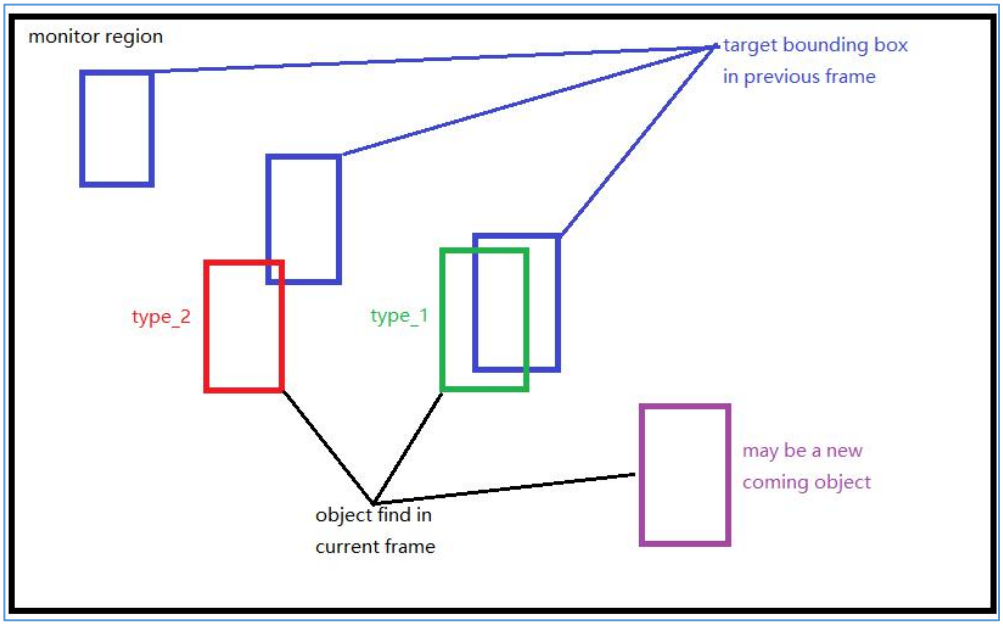


Figure3.3.1 Tracking method

The idea is from the observation that object usually does not move a lot in adjacent frame. Hence for current detected objects, we try to find where is it in last frame. Again we use overlapped ratio to describe how similar two bounding box are. In good case, there should only be one candidate for each object, throw this candidate to MDNet network to compute its target score. If we find more than one candidates, then just throw all of them to network and select the one with highest target score. This is called 'type_1 searching' as shown in Figure 3.3.1.

However, some objects move pretty fast, which may lead to low overlapped ratio or even failing to find any candidate. In this case, we need to run 'type_2 searching', which is shown in Figure 3.3.2. Basically, tracker will perform an exhausting searching in certain region and use top N matched bounding box's mean as target bounding box. (Target score in this case is defined as mean bounding box's target score) Those bounding box are considered as well matched ones according to target score.

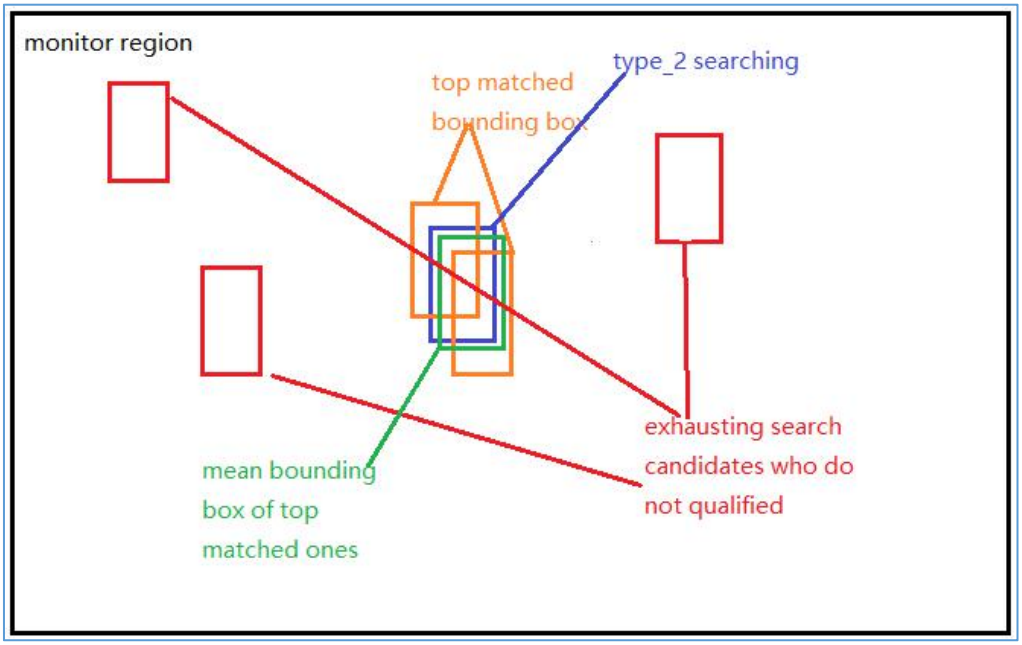


Figure3.3.2 Type_2 searching

Then tracker will ignore candidates with low overlapped ratio and throw top N overlapped candidates to network to get their target score. Finally choose the one with highest target score shown in Figure 3.3.1 type#1. Notice, if this fail to find a target which is good enough, searching region will be expanded.

If target score is higher than threshold, we define this track as 'type_1 tracking', which means we trust this result and will leave it alone. Otherwise we define track as 'type_2 tracking', which means we need to check the outcome later.

3.4 how to deal with coming/leaving and occlusion

Recall there is occlusion and reshow up problem, we solve that via 'double checking module'. This module will be activated after certain frames and only consider 'type_2' tracking. The idea is that if tracker fail to perform a good tracking this frame, we check previous N frames to see this object is successfully tracked before.

If in previous frames there is detection bounding box overlapped with tracking frames, then we consider this is occlusion case and do nothing in double checking module. Else we view this case as target is lost or target is out of monitor region and delete everything of it including model and feature clusters.

After all objects in previous frames settle, we compare object list to see if there is new object coming. If there is, we create its sample pool and model.

3.4 update network

This part is pretty similar to corresponding module in MDNet. If tracking type is

one, we generate positive and negative samples for the object and store them to respective pool. If tracking type is two, we apply a short term update to the network using samples in current pool. This is based on the assumption that bad tracking is due to poor performance of current object sample's model, thus need more training. For every fixed number of frames, we perform a long term update, which will update all valid objects' models using respective sample pool.

3.5 detail implement algorithm

(i) Initialization step

Read first image;
To each object in the image:
    Generate positive samples;
    Generate negative samples;
    Train network;
    Train a simple linear regression model for bounding box;
    Record bounding box;

(ii) Main loop

To rest of the frames:
    To each object in previous frame:
        Does this object still valid?
        If yes:
            Can we find candidates around?
            If yes:
                Selected one with highest target score;
                Target score = highest score;
            Else:
                Target score = -1;
            If target score is higher than T:
                Set tracking type = 1;
                Record bounding box;
            Else:
                Exhaust search in a region;
                Choose mean bounding box from top N matched bounding box;
                Target score = mean bounding box's score;
                Does exhaust search work?
                If not really:
                    Searching region will be expanded;
                Bounding box regression;
                Record bounding box;
            Double check to deal with occlusion issue;
            If success:
                Set success flag;
            If median:
                Set median flag;

If fail:
Check if it is out of scope;
If yes:
Delete it;
Check justif_no flag to see if it is already disappear for N frames;
If yes:
Delete it;
If success:
Generate new samples for object;
If not success:
Train model of current object (short term update)
If reaching long-term update point:
Update all models;
Check if there is any new coming object:
If yes:
Perform initialization step as in (i)

4. Demo application: Traffic flow monitor

We apply MDMONet to demo program: Traffic flow monitor. The program can be divided into three part: Detecting, Tracking and Trace analysis. Given a video, we first convert it to a sequence via FFMPEG. We input the sequence to detecting network, it will detect objects in each frame. Then we run tracking algorithm to compute trace of each object, at the same time trace analysis module runs to check preset rude driving behavior.

4.1 Dataset

Due to lack of appropriate real life data set, we utilize online TPS game grand theft auto V to collect traffic data. When we were at the stage that testing MDMONet without trace analysis, offline mode was run. With help of mods like vehicle loader, trainer and free camera, it is pretty delightful to collect stable traffic flow. In terms of sequence with offensive driving, that is a totally different story. Because mod that can let AI perform rude driving do not exist , we have to collect them in online mode. Sadly, Rockstar ban all cheater mods in online mode, hence we do not have overall control on factor like: which vehicle, what time or what weather. Good observation angles from in offline mode are invalid in online mode. Thus our rude driving behavior detector is more like a demo.

For stable traffic flow recorded in offline mode, there will be three sequences under regular weather, froggy weather and raining. For video with rude driving, there will be two sequence at evening and noon. All of the sequence is recorded @ 2560x1440 resolution and FPS is fixed at 30. The output sequences are at the same size of input.

4.2 Detecting

There are many RCNN based detectors available but their performance on our

dataset vary a lot. Because detecting network provide groundtruth for later tracking, its performance really has a heavy weight. For example, it could be too sensitive that regard one big vehicle as two, or it may be too dump so that detecting results are not stable.

After several trials, we use Single Shot Multi-box Detector[2] as our detecting network, which is raised and implemented by *Liu, Wei and Anguelov, Dragomir and Erhan, Dumitru and Szegedy, Christian and Reed, Scott and Fu, Cheng-Yang and Berg, Alexander C*. Their code is open source and we can train our own dataset with a little bit modification.

After passing sequence to detecting network, it will generate a text file which recorded coordinates of objects detected in each frame. This is one of the input of our tracking network.

Due to fixed detecting region size of SSD, we use a trick to expand the monitor region as shown in Figure 4.2.1.

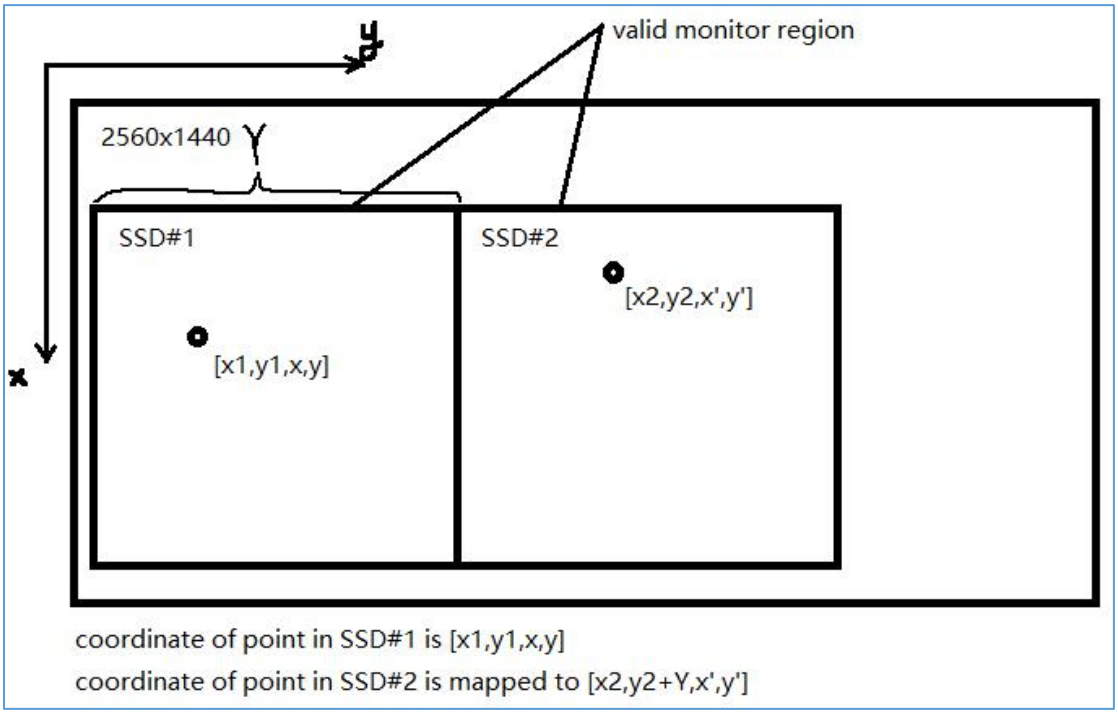


Figure 4.2.1 concatenate two SSD region

4.3 Tracking and Record Trace

MDMONet need two inputs, one is original sequence, another one is detected coordinates in each file as mentioned in 4.2. Our program reserves tuning interface which can let user to change sequence, determine start frame and end frame, dropping condition, output bounding box format and rude driving tolerance standard.

Trace analysis is paralleling to tracking procedure. It is based on bounding box recorded under successful tracking condition. The data structure is (x0,y0,x,y), where (x0,y0) stands for top left corner of bounding box and x and y stand for length at x

and y axis as shown in the Figure 4.3.1. We use center point presented object, which

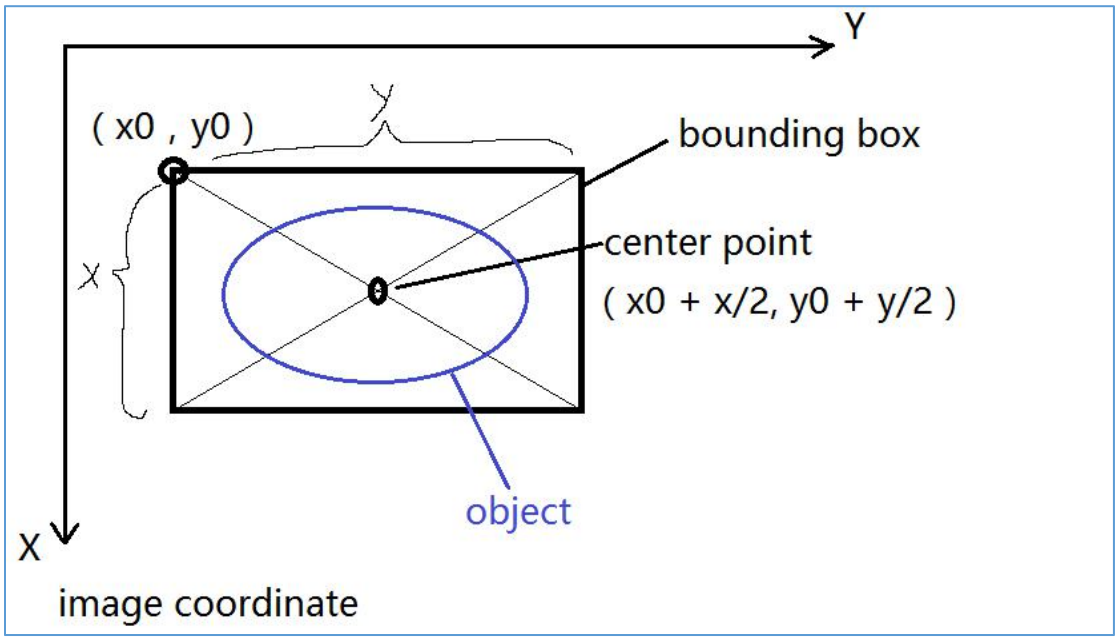


Figure 4.3.1 center point presented object

is more robust than the top left corner presentation we used at first. In experiments, we find that large vehicle like trucks' bounding box change enormously. Size of box and coordinate of top left corner is highly non-linear when trucks are moving along a line stably, but the center presents trace pretty well.

4.4 Trace Analysis

In terms of trace analysis, consider a case that someone drive really fast and keep changing lines rapidly.

We propose two standard: average residual slope (ARS) and average line distance (ALL). Both Average residual slope and average line distance are based on the assumption that normal driven car rarely change lines hence it almost keeps trace as straight line. Those strategies also works for roads that are not straight because rude driving always make higher value at both standards, the difference is just threshold.

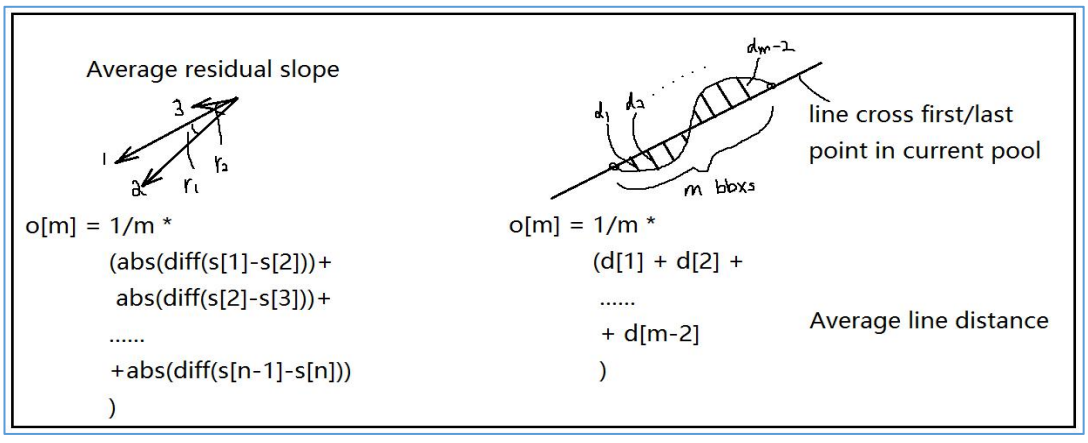


Figure 4.4.1 ARS and ALL

However, in real practice, average residual slope doesn't work. This is because tracked trace is not noiseless. Due to accuracy and scale problem, center of object always have ‘ripple’ even object move along a straight line. Sometimes, ARS can not distinguish noise and real offensive driving, and gap between those two cases are usually smaller than ALL.

We also try other methods, for example run a linear regression with nonlinear base function and check what order does model need to have small enough training error, but that is too time-consuming. In the program we stick with ALL.

5. Experiment Result

We will present following result: detecting result, stable traffic result, rude driving result. Afterward a small comment on those results and possible potential further application will be discussed.

5.1 Detecting Result:

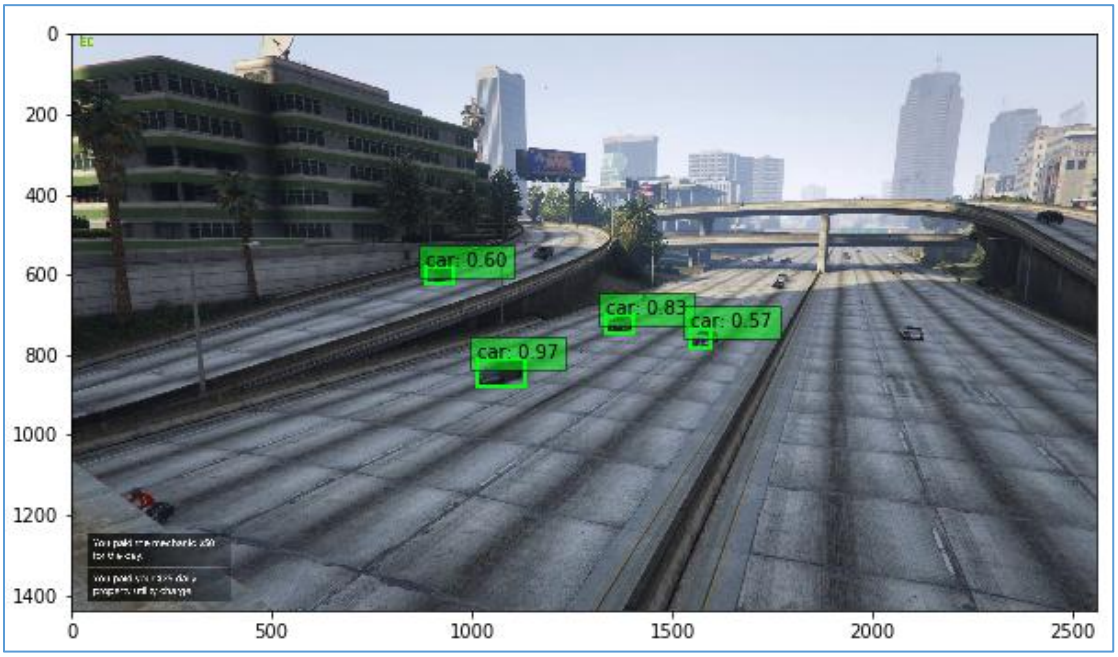


Figure 5.1.1 detecting Result

Corresponding text has following format: [class number, frame index, x0,y0,x,y]. Text will record objects’ coordinates in every frames.

5.2 Stable Traffic flow Result:

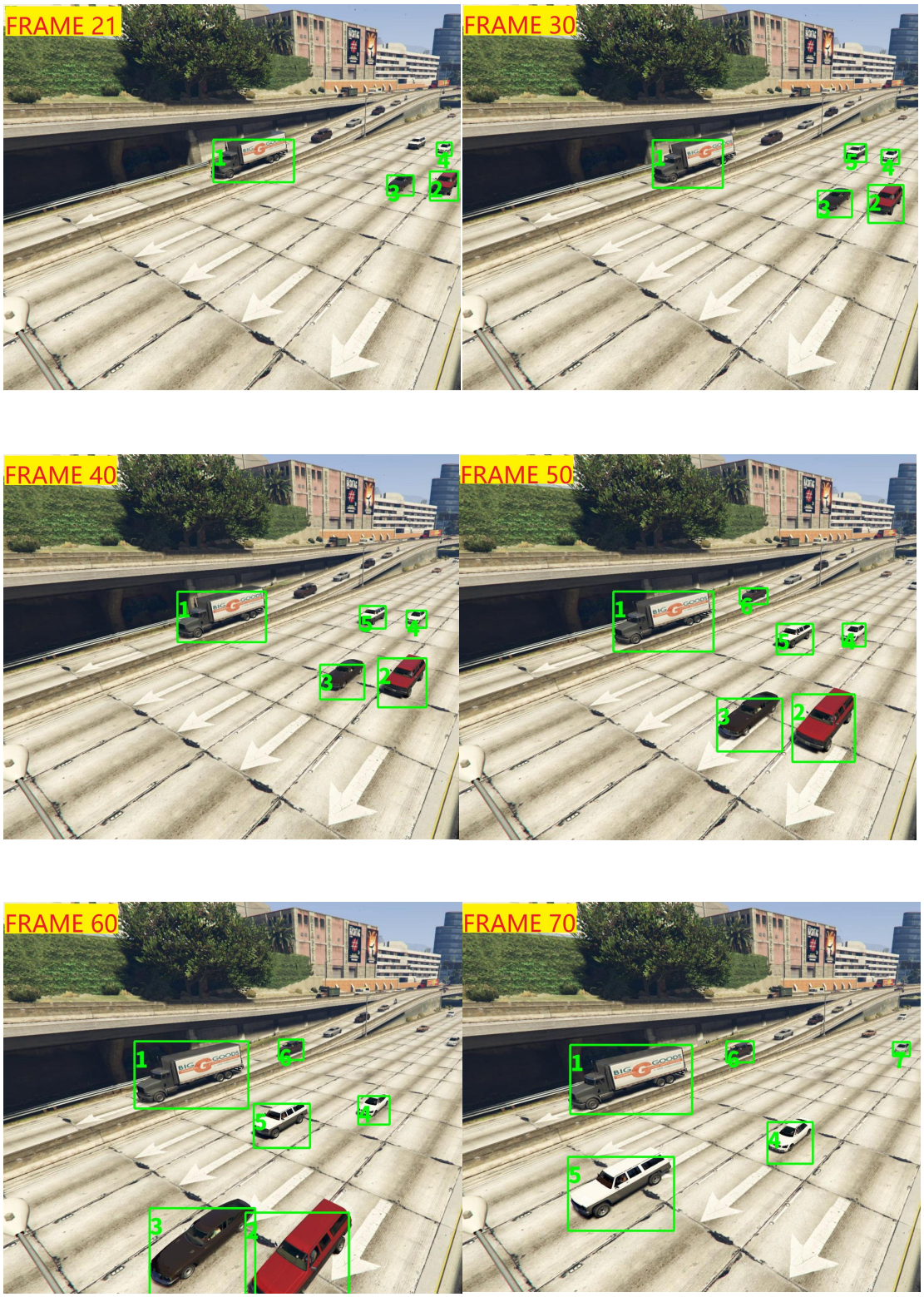


Figure 5.2.1 Stable traffic _ clear weather

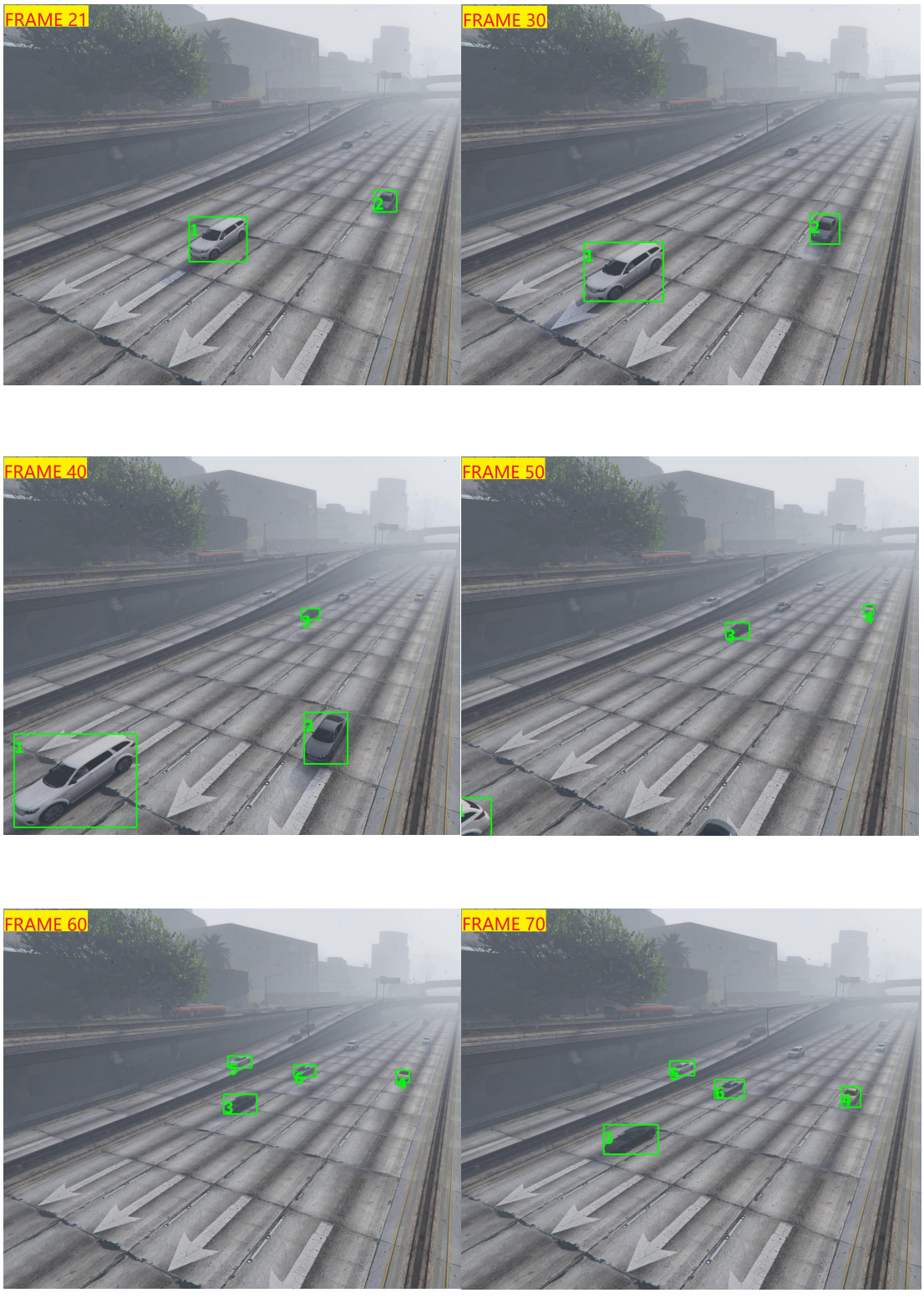


Figure 5.2.2 Stable traffic _ froggy weather

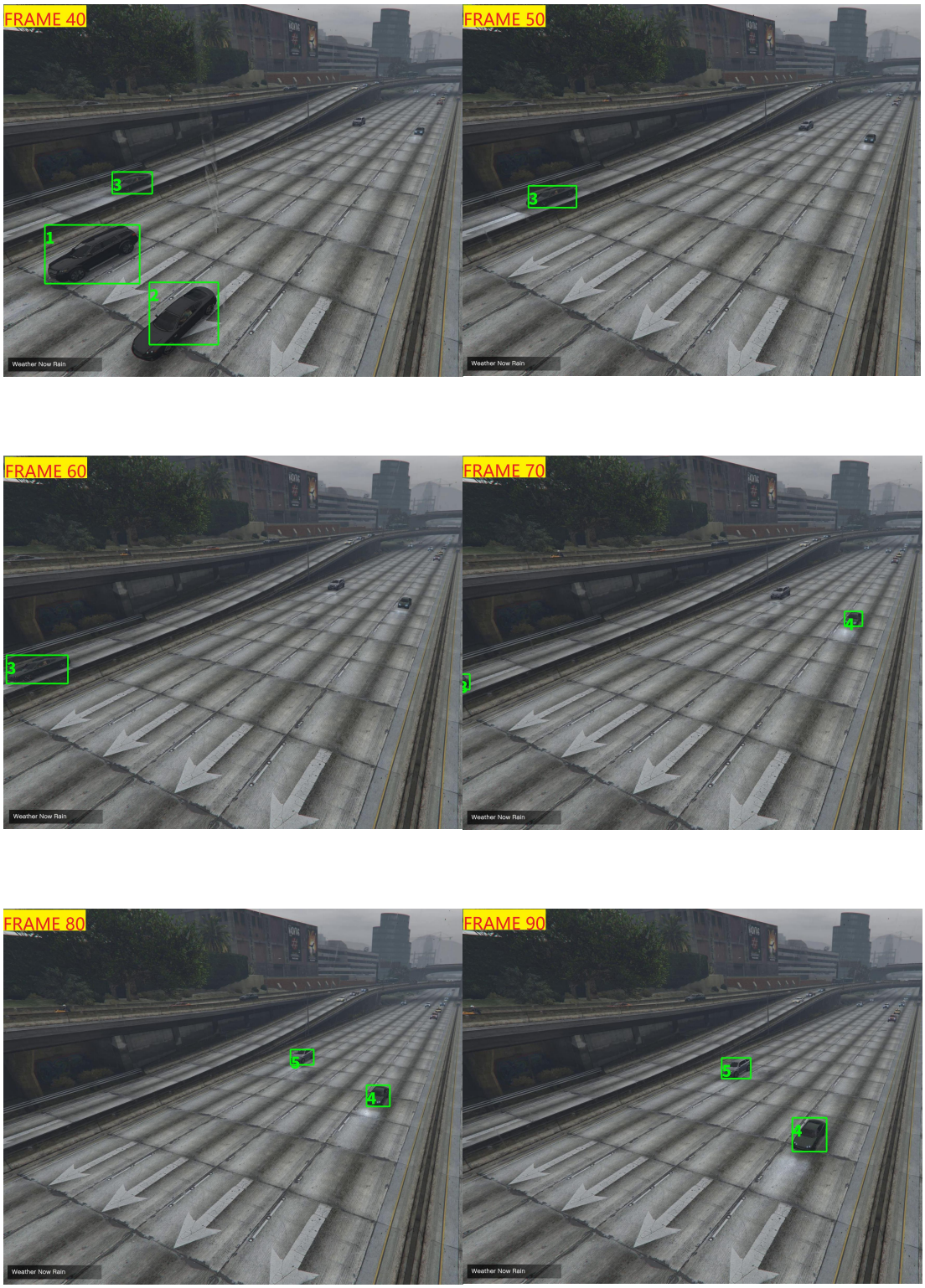


Figure 5.2.3 Stable traffic _ rainy weather

5.3 sequence contain offensive driving

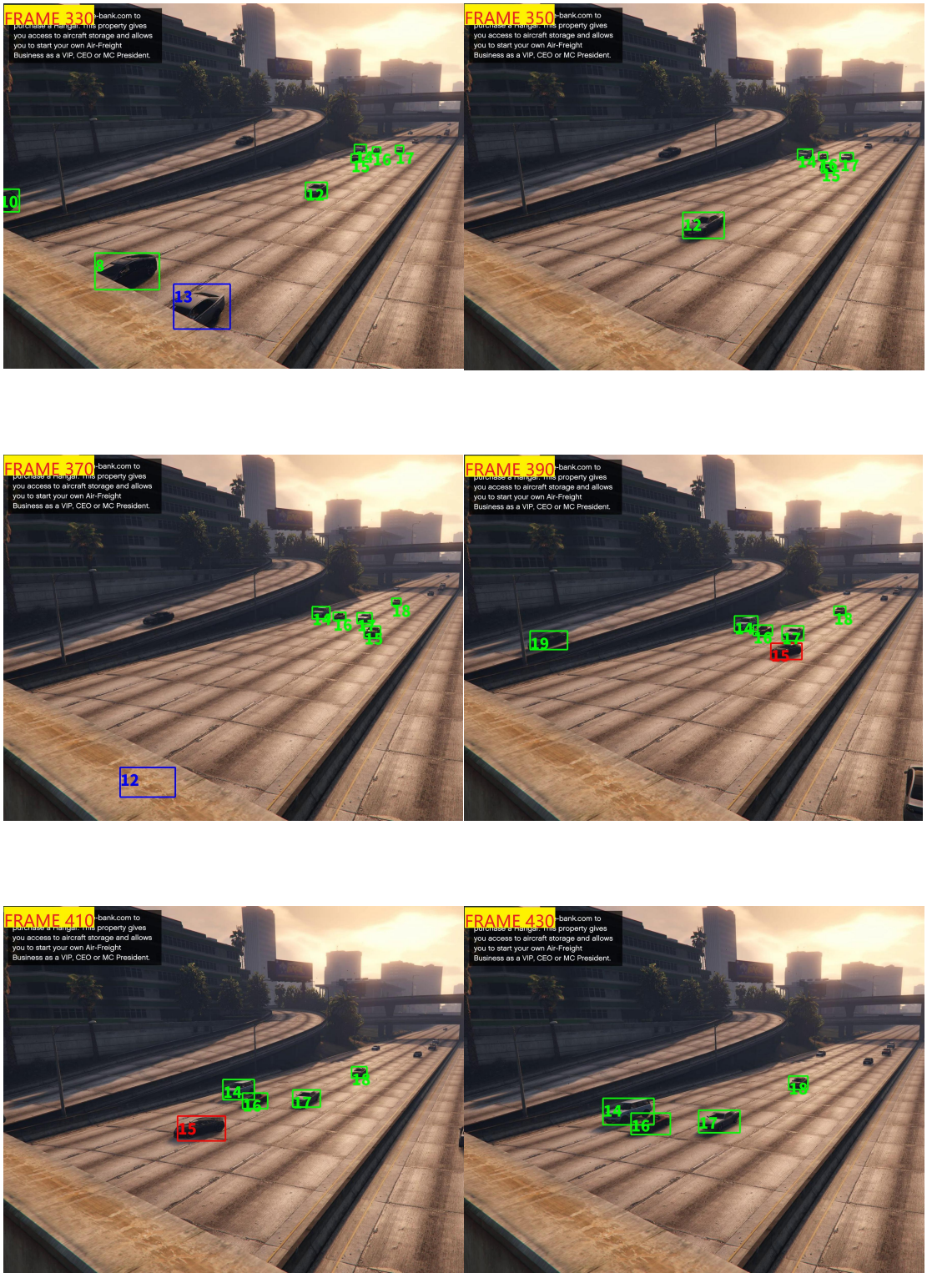


Figure 5.3.1 offensive driving_ evening

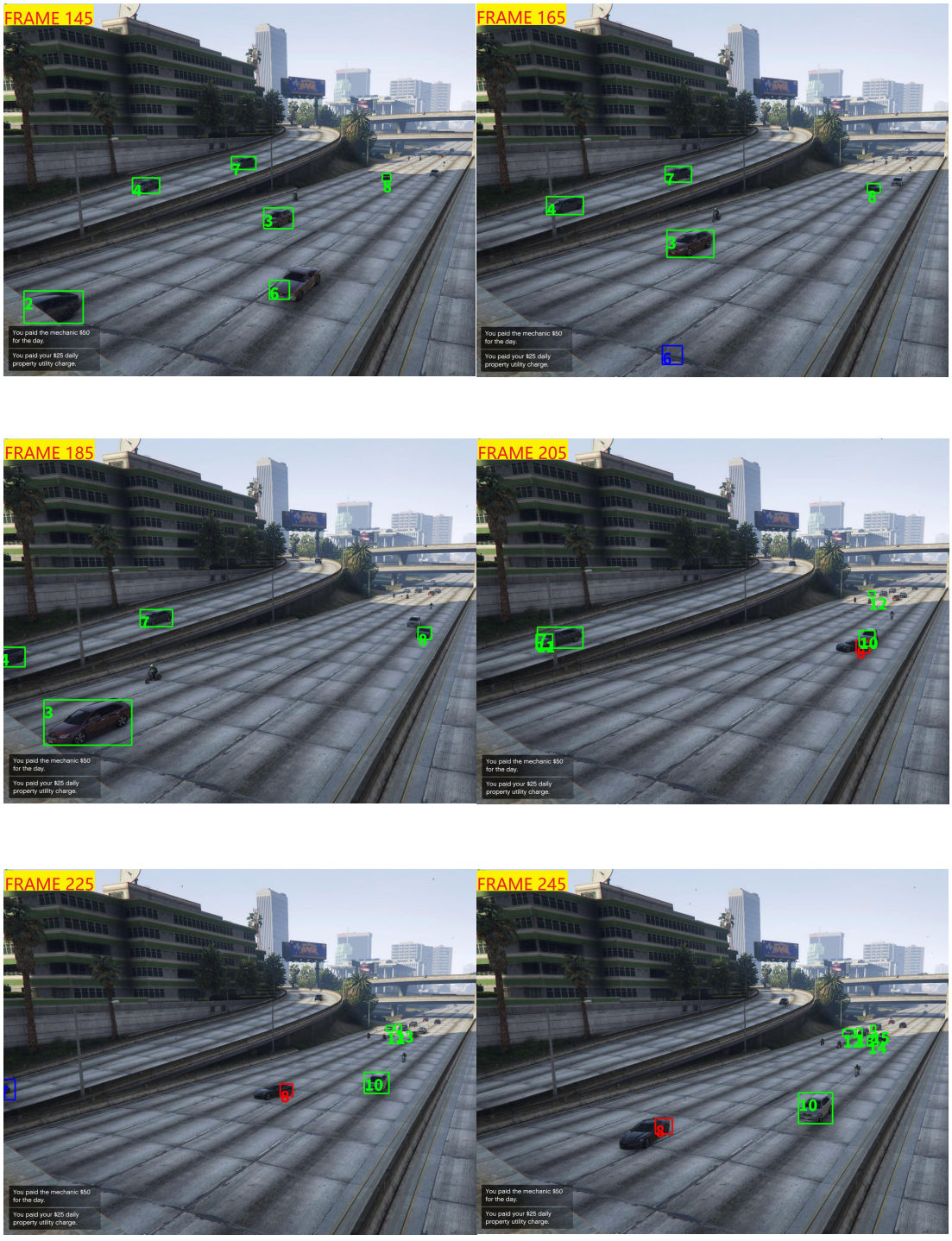


Figure 5.3.2 offensive driving_ morning

6. Conclusion and Further Application

According to experiment result last section, we can draw conclusions that:

- MDMONet update network with previous experience, it collects sample while training, thus it doesn't need a huge amount of labeled data, which is expensive to gather in most case.
- MDMONet is stable and accurate enough to be put into real industry life. (not a single tracking error occurs in the experiment)
- MDMONet is a real-time network, it does not need any further frames' information. Although in our experiment we pass entire sequence to SSD first, that is not necessary.
- MDMONet is hardware friendly. In the whole training process, only three fully-connected layers are trained, which means computation complexity is low. In our program, we delete everything except for status of missing objects, thus memory usage is under control.

There are other applications can be implemented with MDMONet. For instance, our program record each object's index and keep them, thus it can count how many objects pass the monitor region. User just need to set up a camera and shoot frames, MDMONet can handle the rest without any trouble.

Besides, MDMONet manages to record traces of objects, hence user can cascade their own module to take care of different business. In our experiment, we planed to implement an over-speed detection as well. However, due to driving system of GTAV (Top speed is severely restricted and AI drives so fast, afterburner mod is banned so we can not create fast enough vehicle), it is implausible to collect valid data, but this will be pretty easy in real life.