

# Combined Detection and Tracking Impacts on Vehicle Specific Data

*David Zhang*



Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/EECS-2019-24

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-24.html>

May 6, 2019

Copyright © 2019, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

### Acknowledgement

I would like to thank my parents for their lifelong support and encouragement, constantly providing me with the opportunity to strive for my best. I would also like to thank Professor Trevor Darrell, Fisher Yu and HouNing Hu for their guidance and expertise, an invaluable part of my research experience.

# Combined Detection and Tracking Impacts on Vehicle Specific Data

by

David Zhang

A technical report submitted in partial satisfaction of the

requirements for the degree of

Masters of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair

Professor John Canny

Spring 2019

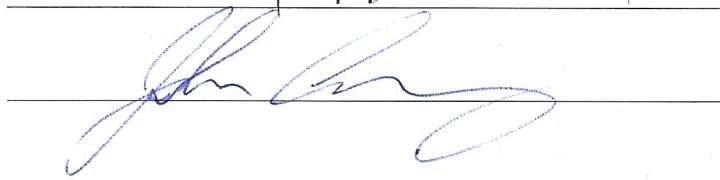
The technical report of David Zhang, titled Combined Detection and Tracking Impacts on Vehicle Specific Data, is approved:

Chair



Date

5/6/19



Date

5/1/19

Date

\_\_\_\_\_

University of California, Berkeley

## Abstract

Combined Detection and Tracking Impacts on Vehicle Specific Data

by

David Zhang

Masters of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

On-the-road vehicle and pedestrian detection for self-driving cars thus far has been a combination of sonar, lidar, and cameras. For cameras, although object detection has improved dramatically in the last 20 years, common algorithms are best suited to handle static images and treat videos as a series of disjoint detections. Sequences across time can hold extra information allowing for improved detection performance as well as the added benefit of tracking objects through time. This research doubles a standard fully convolutional network into a siamese architecture to handle consecutive frame inputs and adds a tracking module to assist in the prediction of frame to frame differences. The architecture is evaluated on both a standard dataset and a custom dataset to test its ability to handle more in-scene complexity.

# Contents

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>ii</b>
<b>List of Tables</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>2</b>
2.1 Object Detection . . . . .	2
2.2 Video Sequence Tracking . . . . .	2
2.3 Vehicle Dataset . . . . .	3
<b>3 Implementation</b>	<b>4</b>
3.1 RFCN . . . . .	4
3.2 Action Tubes . . . . .	5
3.3 Detect to Track . . . . .	6
3.4 Synthetic Dataset . . . . .	8
3.5 Training . . . . .	8
<b>4 Results</b>	<b>9</b>
4.1 Conclusion . . . . .	15
<b>Bibliography</b>	<b>16</b>

# List of Figures

3.1	Position-Sensitive Region of Interest . . . . .	5
3.2	Detect to Track Network Architecture . . . . .	7
4.1	Imagenet Training Loss . . . . .	10
4.2	Imagenet Tracking Loss . . . . .	11
4.3	GTA Training Loss . . . . .	12
4.4	GTA Tracking Loss . . . . .	13
4.5	GTA Tracking Loss Ending . . . . .	14

# List of Tables

4.1	Imagenet Vid AP scores . . . . .	9
4.2	GTA AP scores . . . . .	10



## **Acknowledgments**

I would like to thank my parents for their lifelong support and encouragement, constantly providing me with the opportunity to strive for my best. I would also like to thank Professor Trevor Darrell, Fisher Yu and HouNing Hu for their guidance and expertise, an invaluable part of my research experience.

# Chapter 1

## Introduction

With the increasing computational power of modern day processors, machines are replacing humans in an ever-growing number of tasks. For better or for worse, industrial and academic researchers have pooled resources to hasten this electronic encroachment in a variety of fields. One such field is that of transportation, or more specifically, self-driving cars. The final goal is one day these vehicles will no longer require human focus and interaction, and merely be a temporary respite while shuffling its passengers from point A to point B. This has the potential to bring about many benefits, many of which include improved safety on the road and less time wasted on merely traveling.

In order to facilitate this vehicular development, cars require sensors that allow them to build an understanding of the environment around them. Currently, self-driving systems deployed on the road require a complex variety of sensors involving cameras, sonar, lidar, and GPS. While this can be a fairly robust source of information for the self-driving system, the aggregate complexity leads to a plethora of potential failure points, mechanical and/or electrical, which are undesirable in risk-averse fields like the automotive industry. Ideally failure points would be mitigated or even removed to ensure high uptime of an overall system. Thus, work is being done to offload information reliance from more exotic and expensive sensors like lidar, and mainly relying on cameras to do the major data collection.

While cameras provide excellent information for human analysis, they can be difficult to distill relevant information from. With data represented in large arrays, often contiguous in memory, complex algorithms are required to reduce it to the nice semantic labelling used in intuitive and human-interpretable displays. This paper aims to determine the feasibility of the Detect to Track algorithm in providing semantic labelling in a complex vehicle-centric context comprised of synthetic data.

# Chapter 2

## Related Work

### 2.1 Object Detection

Object detection algorithms, especially those in search of object bounding boxes, notably reference the R-CNN[10] family, which include Fast R-CNN[9] and Faster R-CNN[22]. These algorithms are based on a region proposal network, which selects potentially interesting regions in the target image for the R-CNN network to re-process and classify. The post-processing step generally consumes a large amount of resources since there can be up to several thousand regions, each of which require multiple fully connected layers. Faster R-CNN was able to improve region proposal speed by integrating the region proposal network with the convolutional network, making it “nearly cost-free”. While not necessarily in the same family but based on Faster R-CNN, RFCN[4] significantly improves post-processing speed at a slight cost to accuracy by replacing the fully connected layers with more convolutional layers. The use of convolutional layers allows the per-box computation to be processed simultaneously.

These methods differ from YOLO[21] and SSD[17], which predict bounding boxes at the same time as classification. While the various region-based methods have been able to produce more accurate results, the improvements in speed of these regression-based methods with only a small difference in accuracy can be useful for real-time applications[4].

### 2.2 Video Sequence Tracking

While static image object detection is a necessary first step, tracking individual objects across time can be particularly useful for scene understanding. Some of the previous attempts have used LSTMs[12], correlation filters[1][18], and even post-training fine-tuning for test time predictions[20]. While LSTMs[24] generally perform well in keeping track of data through sequences, the extra bulk of the network at training time is unfavorable for training longer video sequences. In the case of MUSTer[14], the use of handcrafted features limits the scalability of the method. However, Kernelized Correlation Filters provide a fast and

intuitive method for determining movement between frames. This inspires the action tube linking described in section 3.2.

The Siamese style algorithm[1] also heavily inspires this paper, where the network is trained on pairs of subsequent frames. Along with being fully-convolutional, this architecture allows the network to learn similarities between frames to closely track multiple objects. By taking advantage of the time-localized information present in a video sequence, the network is able to learn to predict bounding box changes without relying entirely on a correlation filter.

## 2.3 Vehicle Dataset

Several popular driving datasets currently exist, including KITTI[8], BDD100k[26], Oxford's Robot Car[19], Nuscenes[2], Cityscape[3], etc. KITTI[8] provides multiple sub-datasets for stereo, optical flow, depth, object detection, and more relevant tracking, among others. The KITTI data format is used for Berkeley's BDD100k[26], which provides a hundred thousand videos of images labeled for various kinds of segmentation. Cityscape[3] offers a similar dataset, but on a much smaller scale. The video sequences have only a fraction of images finely labeled, while 80% of the images are coarsely labeled to save labeling resources. Nuscenes[2] is also inspired by KITTI, with a combination of camera, lidar, and radar. However, the full dataset was not released to the public at the time of work.

# Chapter 3

## Implementation

The Detect to Track[7] architecture proposes augmenting a detection network with a tracking module. The detection network is a RFCN[4] network with convolutional trunk like Resnet-101[13]. The detection network is duplicated across two subsequent frames to provide context for the tracking module. The tracking module hooks into the convolutional trunk by cross-correlating intermediate layers and using the result as inputs to a convolutional layer. The cross-correlation indicates areas of high similarity between the legs of the network, a useful feature for getting the movement of an object. The robustness of this tracking module is tested on self-driving data in the form of the GTA dataset[15].

### 3.1 RFCN

The fully convolutional architecture used is the Region-based Fully Convolutional Network[4]. In order to improve wall time performance, the Faster-RCNN[22] architecture is augmented by a Position Sensitive Roi Pooling (PSRoi) layer[4] that replaces the multiple fully connected layers. The trunk network is typically a ResNet-101[13], due to the natural association to use a fully convolutional trunk. The PSRoi layer allows the network to learn translation variant features in respect to object detection, while maintaining the translation invariant features that allow for classification of either the entire image or roi's. Visualized in figure 3.1, this layer works by first adding a convolutional layer that creates  $k \times k$  score maps corresponding to various high level features to be pooled. Each score map also encodes translation variant information due to the pooling layer, which adjusts its relevant region to a fractional portion. The pooling layer then uses the regions of interest to select the area of the score map to average. The  $k \times k$  regions are then summed to calculate a vote for the region. With a region properly centered and scaled to an object of interest, the vote will be activated relatively high. A misaligned region likewise will have a properly low activation. This final pooling layer is the only repeated region specific layer, reducing region based computation significantly without negatively impacting classification performance. The authors report results matching and often beating the naive Faster-Rcnn[22] implementation on Voc07[5]

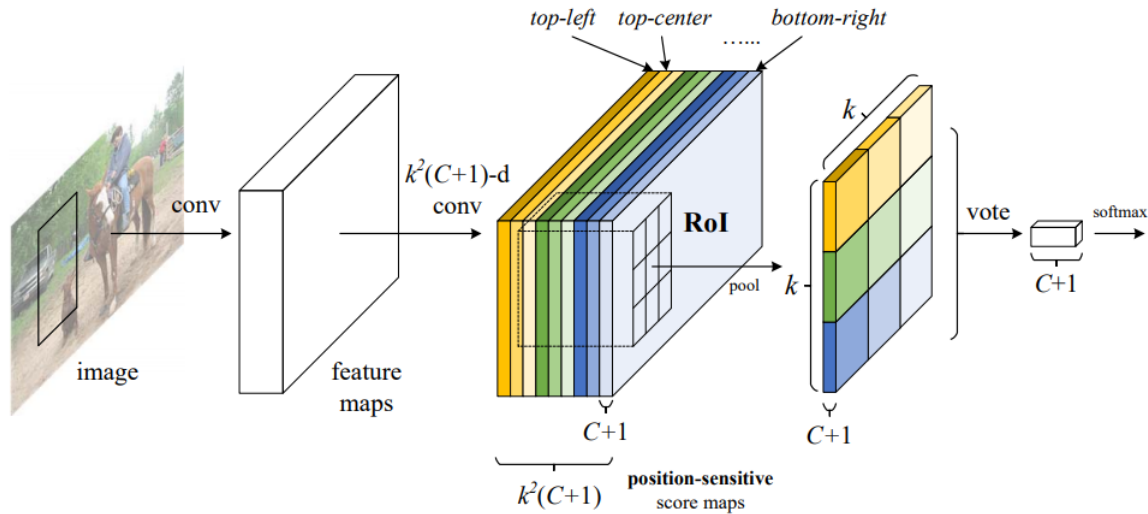


Figure 3.1: Position-Sensitive Region of Interest Pooling. Graphic borrowed from Dai et al[4]

and Voc12[6], although Faster-Rcnn improved by adding iterative box regression, context, and multi-scale testing[4] is able to marginally edge out RFCN in all reported metrics. In terms of runtime, however, RFCN is several time faster than both Faster-Rcnn implementations. The naive implementation averages .42 seconds per image while Faster-Rcnn +++ takes over 3 seconds. RFCN takes a mere .17 seconds to achieve respectable performance, even in comparison to Faster-Rcnn +++. With video datasets orders of magnitude larger than Voc07 and even COCO, the wall time performance is a greatly appreciated property of RFCN. Also the significantly improved test performance is critical for real time applications like self driving vehicles, making this a good choice over other options with better classification performance.

## 3.2 Action Tubes

In tracking objects, having temporal context provides valuable information in narrowing down possible positions. In Finding Action Tubes[11], the authors use two action based datasets, UCF Sports[25] and J-HMDB[16], to test their frame linking algorithm. First, for each frame, localized bounding boxes are extracted with a cnn, and scores for each possible action are predicted. The list of actions are then filtered by optimizing the iou between each consecutive frame. A single path is extracted at a time and the process is repeated until there are not enough bounding boxes to complete a full path. Each path receives a score based on the average of each bounding box score used to create the path. This score averaging has the potential to boost true positive bounding boxes that are either low score or beaten out by other false positive predictions. They reason each frame will have detections reasonably

close to one another and the correct action will be dominant throughout the video sequence. As a result they achieve significant gains over other published results.

However, this implementation has some shortcomings in cases of multiple objects in the scene. The authors encountered significant accuracy disparity in the case of the running action, where multiple people are running simultaneously. Although they report reasonable performance in individual frames, the action tube linking is unable to capture the slightly more complex motion of multiple people in combination with the camera’s ego-motion. They state the use action tubes is for the purpose of linking and action prediction, not object localization, which they consider out of scope for their work. However, this paper attempts to do just this but with a slightly modified implementation to hopefully remedy those shortcomings.

### 3.3 Detect to Track

The Detect to track paper[7] takes the two previously mentioned improvements and mixes them into a single model. They train and test on the Imagenet Video dataset, comprised of over 3000 video sequences, significantly larger than the other sequence based datasets. However, unlike the main Imagenet Detection dataset, Vid uses a 30 class subset of the total 20000 classes. This reduction in classes could hide true performance on tasks with a significant amount of classes, but in this case as well as the gta dataset, the focus is on wide generalization per class rather than a large number of classes.

For the model architecture, the base resnet-101[13] model was used for its balanced performance in speed and accuracy. From there the resnet-101 trunk is fed into a pair of convolutional layers with  $7 \times 7$  layers that correspond to the  $7 \times 7$  psroi[4] pooling area. These two pooling layers lead to an average pooling layer that ultimately predicts the classification and bounding box scores. The two scores are supported by separate convolutional layers in order to allow the bounding box detection to be class-agnostic, generalizing on all features that indicate the existence of any relevant object in the image. Apart from the standard RFCN architecture, the outputs of the conv3, conv4, and conv5 layers of the resnet[13] trunk are cross correlated between each frame to find translational context. The cross correlation should have high activation where the object matches in the two images, however this can be affected by scene obstructions as well as the object disappearing from the camera view entirely. In the general case, the high activation will help provide the context necessary to predict the localization disparity between the two time frames, later used to create action tubes. The cross correlation result is first concatenated with the outputs of the bounding box convolution layers for each half of the network. These aggregated features are then fed into another convolution layer that again has  $7 \times 7$  score maps for the psroi pooling layer. The predicted output is ultimately the per-box translation and scaling, represented by the delta in center, height and width.

In total, the model will give 600 bounding boxes, 300 for each image. Each box is given a prediction score for each class as well as the tracking delta. During testing the bounding

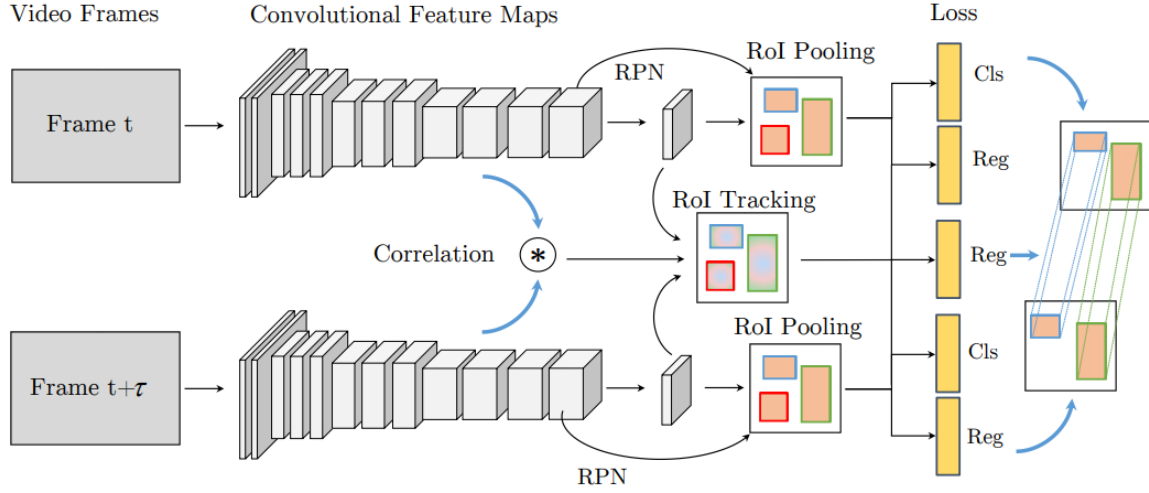


Figure 3.2: Detect to Track Network Architecture. Visualization borrowed from Feichtenhofer et al[7]

boxes get linked into action tubes similarly to the action tube paper. However, instead of merely maximizing the iou of bounding box pairs, the tracking delta is first applied to the first box to match the second box. For each class, tracking scores are calculated using equation 3.1 where  $D_{i,c}^t$  is the detection score for bounding box  $i$  at time  $t$  for class  $c$ .

$$s_c(D_{i,c}^t, D_{j,c}^{t+\tau}, T^{t,t+\tau}) = p_{i,c}^t + p_{j,c}^{t+\tau} + \psi(D_i^t, D_j^{t+\tau}, T^{t,t+\tau}) \quad (3.1)$$

$$\psi(D_i^t, D_j^{t+\tau}, T^{t,t+\tau}) = \begin{cases} 1 & \text{IOU}(D_i^t + T^{t,t+\tau}, D_j^{t+\tau}) > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \quad (3.2)$$

$$P_c = \operatorname{argmax}_P \frac{1}{\tau} \sum_{t=0}^{\tau} s_c(D_i^t, D_j^{t+\tau}, T^{t,t+\tau}) \quad (3.3)$$

$T$  is the tracking delta between frame  $t$  and  $t + \tau$ . Optimal path  $P_c$  for class  $c$  is the sequence of detections with the largest average score. Equation 3.2 ignores class scores to allow for class agnostic bounding box tracking. With a properly trained tracking module, the tracking delta will effectively move the old bounding box over the new bounding box to maximize the iou even over large movements. In this way, the tracking performance of the action tube linking is consistent across sequences with multiple objects of interest. Individual bounding boxes will be guided to the corresponding box in subsequent frames. With the adjusted bounding boxes, the Viterbi algorithm is run on equation 3.3 to find the paths of highest score.

The tracking module consists of a correlation layer, which cross correlates the outputs at various layers of the convolutional trunk, a conv layer, and a psroi layer. While this is a relatively simple design, the lack of complexity also presents itself as drawback in the network architecture. The instance matching between frames can still be an issue. Similar



to the runner matching issues seen in the action tube paper, the largely visually similar cars in a scene will cause the correlation layer to have high outputs for incorrect matches. This problem is compounded by the use of a synthetic dataset, where scene object objects lack subtle differences that can help differentiate similar looking objects. This can be seen in the relatively high tracking loss seen throughout the end of training. Compared to the model trained on imagenet, the model trained on the gta dataset struggles to lower the loss beyond 1.0 and accurately predict the correct tracking delta.

### 3.4 Synthetic Dataset

To test this architecture on vehicle data, a video dataset from the viewpoint of a vehicle on the road would be ideal. In this case, imagenet would not be sufficient, whereas something like KITTI's tracking dataset would be better. However, in the end, a synthetic dataset created from a video game Grand Theft Auto V(GTAV) was used. The GTAV dataset[15] not only includes orders of magnitude more data than the KITTI dataset, but also uses the game engine to accurately define pixel perfect bounding boxes. In terms of realism, the GTAV dataset is subjectively looks reasonably lifelike and should be a relevant test for the performance of the tracking system.

Due to constraints on hard disk space, the entire GTAV dataset was not used. 40% of the training data was randomly selected, but the entire validation set was kept for evaluation repeatability.

### 3.5 Training

The model is trained and tested on both the Imagenet[23] Vid and the gta dataset. For Imagenet, the dataset was reduced by downsampling each video sequence to 10 evenly spaced frames. While this results in longer effective intervals than the DtoT paper attempted, this forces the tracking module to train on more complex movements. The model started with a rfcn model pretrained on detection imagenet, and training continued until the loss plateaued even after gamma decay. This allowed the training to focus on the tracking module since the detection elements were mostly trained. For the gta dataset, full sequences were used to simulate a running system. The model was trained in a similar fashion, but only the resnet-101 trunk was pretrained, allowing for fine tuning on the synthetic domain.

# Chapter 4

## Results

While the training process showed promise, the final results indicate issues in the implementation. When fine tuned in the Imagenet dataset, seen in figure 4.1, the loss has minimal change across the 10 epochs of training. In figure 4.2 the tracking loss makes no significant improvement across the fine tuning process. However, the value being generally small should result in decent tracking deltas. The same cannot be said for the model trained on the GTA dataset. The loss figure 4.3 shows a general trend of improvement across the 20 epochs of training. Once the overall loss reaches around 1, the loss stagnates, matching the tracking loss in both figures 4.4 and 4.5.

	Airplane	Antelope	Bear	Bicycle	Bird	Bus	Car	Cattle	Dog	Domestic Cat	Elephant	Fox	Giant Panda	Hamster	Horse	Lion
w/o Paths	.7120	.7176	.6858	.4946	.5128	.5800	.5278	.5597	.6097	.6168	.6439	.7460	.7069	.7629	.6746	.4021
w/ Paths	.6838	.6729	.6878	.4494	.5215	.5443	.4739	.6108	.6196	.5943	.6386	.7588	.6943	.7879	.6097	.4526
	Lizard	Monkey	Motorcycle	Rabbit	Red Panda	Sheep	Snake	Squirrel	Tiger	Train	Turtle	Watercraft	Whale	Zebra	mAP	
w/o Paths	.5915	.4603	.7222	.5218	.6868	.3147	.6481	.2883	.6752	.7418	.5981	.6318	.5365	.7205	.6030	
w/ Paths	.5959	.3630	.7295	.5389	.6050	.3813	.5482	.2405	.6341	.6615	.5914	.4981	.4172	.7835	.5796	

Table 4.1: Values presented are from evaluating on the Imagenet Vid Validation set. The first row values are from evaluating score without using action linking, whereas the second applies action linking per video snippet. For the most part the results do not indicate a significant difference between the two evaluations, although adding the pathing makes the mAP slightly worse. This is likely attributed to an implementation bug in the action linking. A few classes show slightly improved scores, but scores are slightly lower for the most part.

	Car	Truck	mAP
w/o Paths	.7336	.6163	.6750
w/ Paths	.4007	.3038	.3523

Table 4.2: Similar to table 4.1, values are from evaluating on the GTA validation set. The dataset contained only car and truck detections, so all other classes are ignored. Unlike that of the previous table, the scores here take a significant hit when action linking is applied. While the same implementation issues must have contributed to the stark contrast, figure 4.4 shows the large tracking loss by the end of training.

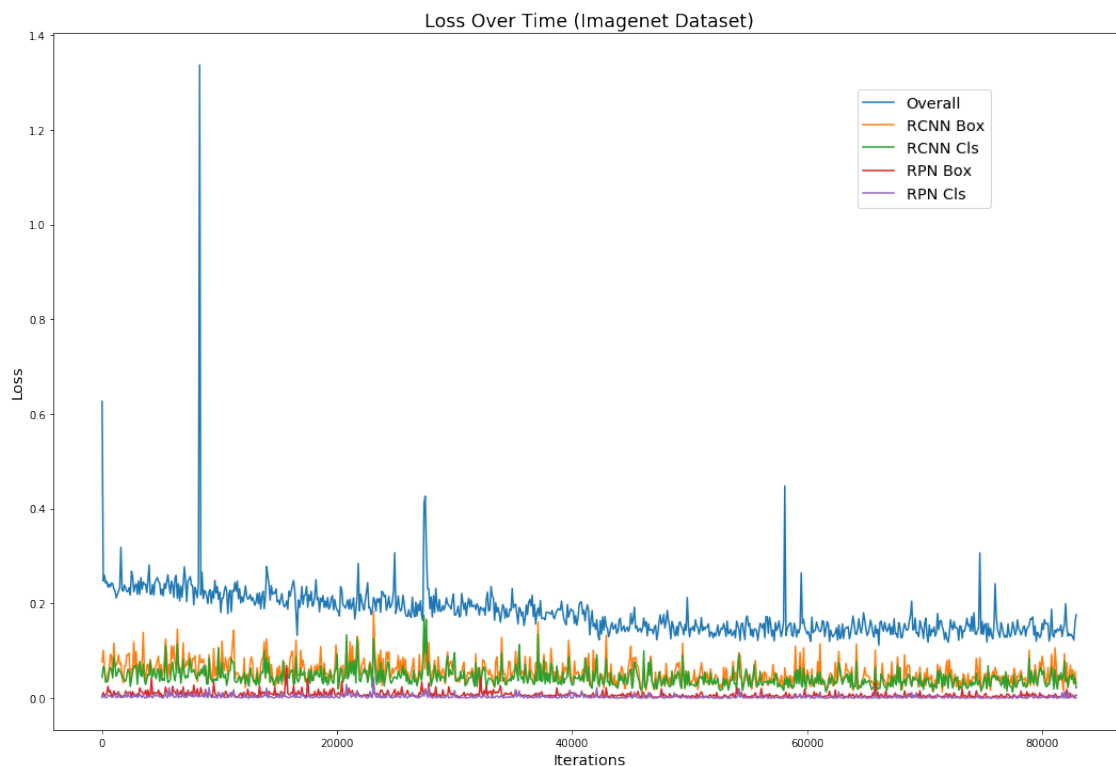


Figure 4.1: Imagenet Training Loss Over 10 Epochs. Tracking loss(4.2) is visualized on a separate graph due to its erratic values. Here the training loss changes little over 10 epochs, dropping from the initial 0.25 to 0.15. Separate loss values for both bounding box and classification are graphed for both the RPN and overall network. Very little change can be observed for those, so the majority of the change must have come from fine tuning the tracking loss.

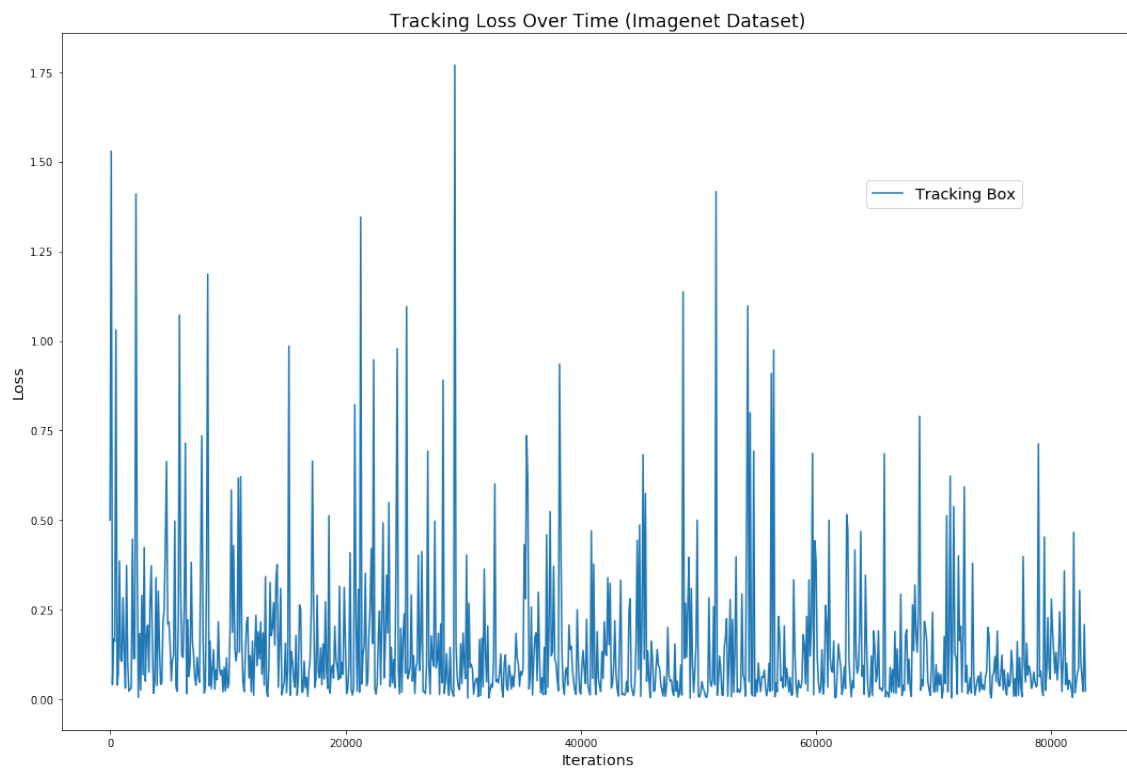


Figure 4.2: Tracking loss for the imagenet dataset. For the most part, the loss is erratic and highly varied, with only a slight downward trend based on the peaky outliers. Many of the data points are very close to 0, likely owing to the frames with minimal or no movement. This illustrates the tracking module’s difficulty in accurately predicting the tracking deltas for all situations.

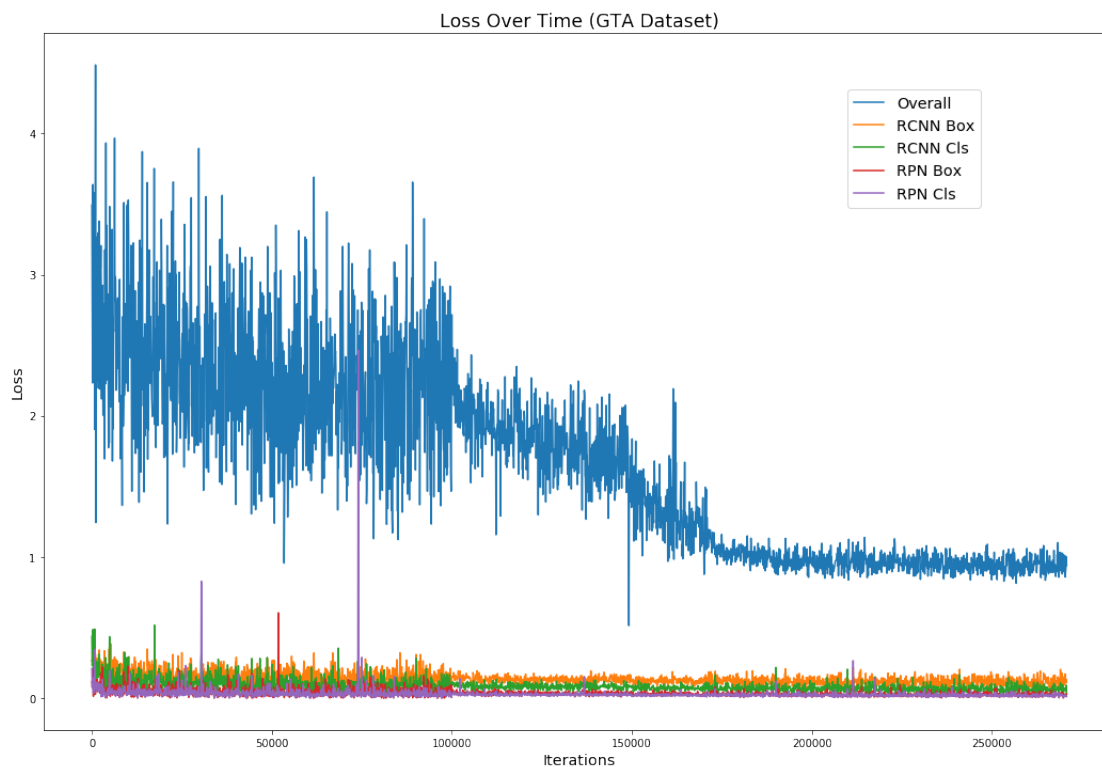


Figure 4.3: Training Loss for the GTA dataset over 20 epochs. Here the loss shows significant change as the learning rate decays. Initially the learning rate of  $1e^{-3}$  is somewhat higher than optimal, leading to the large variances up to 100000 iterations. Once the learning rate decayed to  $1e^{-4}$  at that point, the loss dropped until a plateau around 175000 iterations. The individual loss values are again plotted to show a lack of contribution to the high loss.

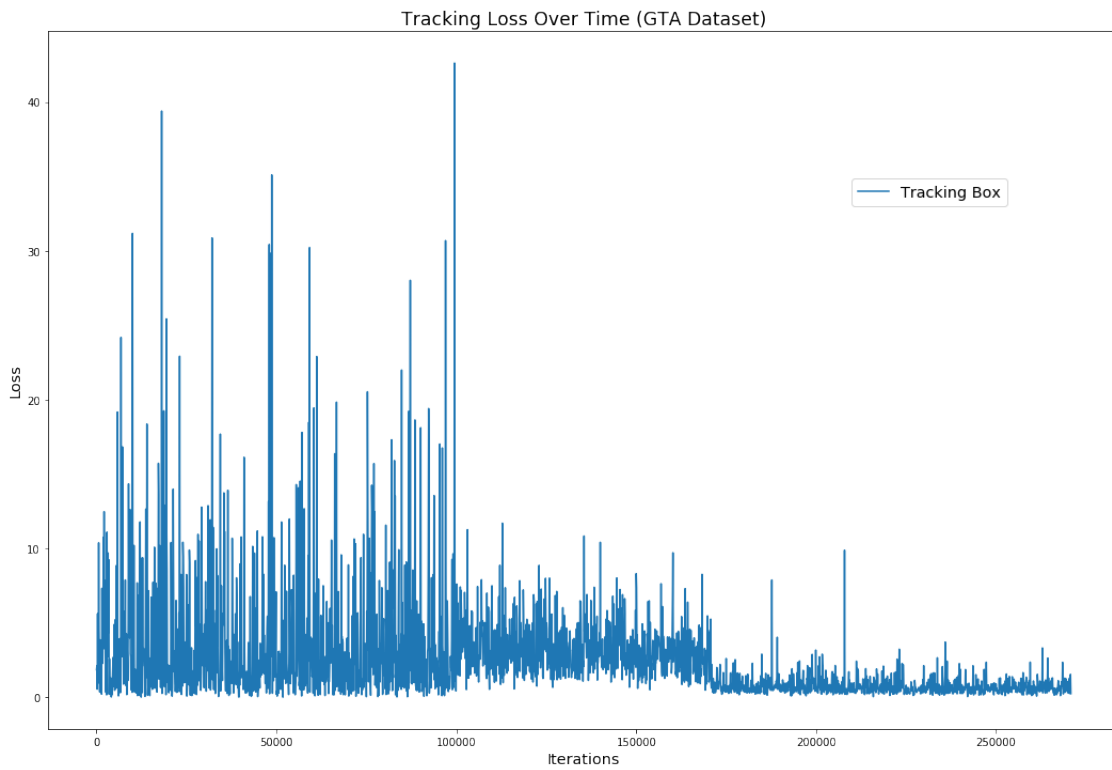


Figure 4.4: Tracking loss for the GTA dataset over 20 epochs. Here the tracking loss shows a large amount of change, dropping its erratic loss of up to 40 down to an average of 1. However, figure 4.5 shows the plateau still exhibits a similar loss to the fine tuned imagenet.

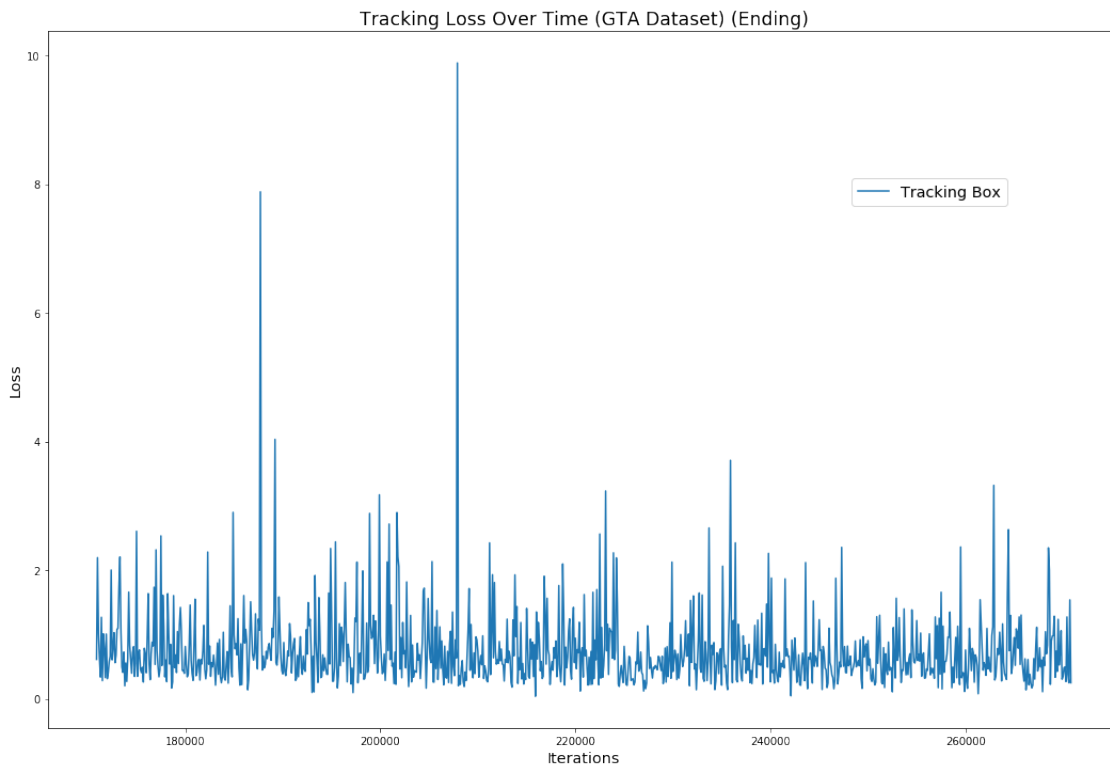


Figure 4.5: Tracking loss for the GTA dataset after 175000 iterations. The tracking loss still has a relatively large variance, with no sign of decreasing. This is likely due to the tracking module’s lack of complexity to capture the tracking deltas in a wide variety of situations and with multiple objects in frame.

## 4.1 Conclusion

The Siamese RFCN network with tracking module is a simple and fast design for multi-frame object detection. The tracking module has the potential to support the convolutional network with information across time, but more work is needed to tailor the depth of the module for more complex datasets. Comparing figures 4.2 and 4.4 indicate these shortcomings, with a significant gap between the two datasets. Especially With the GTA dataset not including more real-world complexity such as pedestrians or road signs, the one layer tracking module is bottle-necking the performance of the network as a whole. Potential solutions include forgoing the PSROI for that module and reverting back to a fully connected network, adding more inputs for the tracking module beyond the correlation features, and simply adding more layers. Using a fully connected network can potentially improve the performance of the tracking module in the same way the faster-rcnn network outperforms rfcn, likely due to the higher complexity of the fully connected layers. Adding more inputs can improve context that the tracking module has to determine individual bounding boxes and pick out the correctly tracked target out of a large number of them. Simply increasing the layer count in the tracking module is the most dubious solution, since the benefit of the PSROI layer is allowing a single convolutional layer to approximate multiple fully connected layers. However, it still has the possibility of improving the complexity needed to capture the variety of situations in the GTA dataset.



# Bibliography

- [1] Luca Bertinetto et al. “Fully-Convolutional Siamese Networks for Object Tracking”. In: *CoRR* abs/1606.09549 (2016). arXiv: 1606.09549. URL: <http://arxiv.org/abs/1606.09549>.
- [2] Holger Caesar et al. “nuScenes: A multimodal dataset for autonomous driving”. In: *arXiv preprint arXiv:1903.11027* (2019).
- [3] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [4] Jifeng Dai et al. “R-FCN: Object Detection via Region-based Fully Convolutional Networks”. In: *CoRR* abs/1605.06409 (2016). arXiv: 1605.06409. URL: <http://arxiv.org/abs/1605.06409>.
- [5] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. URL: <http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html>.
- [6] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. URL: <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [7] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Detect to Track and Track to Detect”. In: *CoRR* abs/1710.03958 (2017). arXiv: 1710.03958. URL: <http://arxiv.org/abs/1710.03958>.
- [8] Andreas Geiger et al. “Vision meets Robotics: The KITTI Dataset”. In: *International Journal of Robotics Research (IJRR)* (2013).
- [9] Ross B. Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: <http://arxiv.org/abs/1504.08083>.
- [10] Ross B. Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: <http://arxiv.org/abs/1311.2524>.

- [11] Georgia Gkioxari and Jitendra Malik. “Finding action tubes”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, pp. 759–768. DOI: 10.1109/CVPR.2015.7298676. URL: <https://doi.org/10.1109/CVPR.2015.7298676>.
- [12] Daniel Gordon, Ali Farhadi, and Dieter Fox. “Re3 : Real-Time Recurrent Regression Networks for Object Tracking”. In: *CoRR* abs/1705.06368 (2017). arXiv: 1705.06368. URL: <http://arxiv.org/abs/1705.06368>.
- [13] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: <http://arxiv.org/abs/1512.03385>.
- [14] Zhibin Hong et al. “MULTi-Store Tracker (MUSTer): A cognitive psychology inspired approach to object tracking”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, pp. 749–758. DOI: 10.1109/CVPR.2015.7298675. URL: <https://doi.org/10.1109/CVPR.2015.7298675>.
- [15] Hou-Ning Hu et al. “Joint Monocular 3D Vehicle Detection and Tracking”. In: *CoRR* abs/1811.10742 (2018). arXiv: 1811.10742. URL: <http://arxiv.org/abs/1811.10742>.
- [16] Hueihan Jhuang et al. “Towards understanding action recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 3192–3199.
- [17] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: <http://arxiv.org/abs/1512.02325>.
- [18] Chao Ma et al. “Hierarchical Convolutional Features for Visual Tracking”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, pp. 3074–3082. DOI: 10.1109/ICCV.2015.352. URL: <https://doi.org/10.1109/ICCV.2015.352>.
- [19] Will Maddern et al. “1 Year, 1000km: The Oxford RobotCar Dataset”. In: *The International Journal of Robotics Research (IJRR)* 36.1 (2017), pp. 3–15. DOI: 10.1177/0278364916679498. eprint: <http://ijr.sagepub.com/content/early/2016/11/28/0278364916679498.full.pdf+html>. URL: <http://dx.doi.org/10.1177/0278364916679498>.
- [20] Hyeonseob Nam and Bohyung Han. “Learning Multi-Domain Convolutional Neural Networks for Visual Tracking”. In: *CoRR* abs/1510.07945 (2015). arXiv: 1510.07945. URL: <http://arxiv.org/abs/1510.07945>.
- [21] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: <http://arxiv.org/abs/1612.08242>.
- [22] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: <http://arxiv.org/abs/1506.01497>.

- [23] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.
- [24] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network”. In: *CoRR* abs/1808.03314 (2018). arXiv: 1808.03314. URL: <http://arxiv.org/abs/1808.03314>.
- [25] Khurram Soomro and Amir R Zamir. “Action recognition in realistic sports videos”. In: *Computer vision in sports*. Springer, 2014, pp. 181–208.
- [26] Huazhe Xu et al. “End-to-end Learning of Driving Models from Large-scale Video Datasets”. In: *CoRR* abs/1612.01079 (2016). arXiv: 1612.01079. URL: <http://arxiv.org/abs/1612.01079>.