City-Scale Vehicle Tracking and Traffic Flow Estimation using Low Frame-Rate Traffic Cameras

Peter Wei
wei.peter@columbia.edu
Columbia University

Haocong Shi
hs2991@columbia.edu
Columbia University

Jiaying Yang
jy2857@columbia.edu
Columbia University

Jingyi Qian
jq2260@columbia.edu
Columbia University

Yinan Ji
yj2470@columbia.edu
Columbia University

Xiaofan Jiang
jiang@ee.columbia.edu
Columbia University

ABSTRACT

Vehicle flow estimation has many potential smart cities and transportation applications. Many cities have existing camera networks which broadcast image feeds; however, the resolution and frame-rate are too low for existing computer vision algorithms to accurately estimate flow. In this work, we present a computer vision and deep learning framework for vehicle tracking. We demonstrate a novel tracking pipeline which enables accurate flow estimates in a range of environments under low resolution and frame-rate constraints. We demonstrate that our system is able to track vehicles in New York City's traffic camera video feeds at 1 Hz or lower frame-rate, and produces higher traffic flow accuracy than popular open source tracking frameworks.

CCS CONCEPTS

• **Computing methodologies** → **Tracking**; • **Software and its engineering** → *Real-time systems software.*

KEYWORDS

computer vision, real-time systems, traffic flow

ACM Reference Format:
Peter Wei, Haocong Shi, Jiaying Yang, Jingyi Qian, Yinan Ji, and Xiaofan Jiang. 2019. City-Scale Vehicle Tracking and Traffic Flow Estimation using Low Frame-Rate Traffic Cameras. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and the 2019 International Symposium on Wearable Computers (UbiComp/ISWC '19 Adjunct), September 9–13, 2019, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3341162.3349336

UbiComp/ISWC '19 Adjunct, September 9–13, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6869-8/19/09…$15.00
https://doi.org/10.1145/3341162.3349336

1 INTRODUCTION

Vehicle tracking and traffic flow estimation are important problems which have many applications in smart transportation and smart cities. Most notably, vehicle flow plays key roles in intelligent transportation systems (ITS), including providing real-time traffic information for analysis and/or intervention, congestion avoidance, route planning, and long-term historical information for urban planning. Additionally, vehicle tracking can be used for novel real-time city-wide applications such as pedestrian [5, 24] and vehicular safety.

Many cities have deployed camera networks at intersections for the purpose of monitoring and analyzing traffic, and to provide data for interested groups (such as developers). However, due to bandwidth and storage limitations, real-time full resolution video streams are often unavailable. For example, New York City's Department of Transportation has made video feeds from all 752 traffic cameras available to the public [19]. However, videos are transmitted at extremely low frame-rate (e.g. 1 image per second or even longer), and at a resolution of 352x240. Since many current methods for tracking vehicles rely on high frame-rate and high resolution images, these data sources lead to high error rates for current methods, and are not usable.

We present a real-time vehicle tracking and traffic flow system ***LFTSys*** (Low Frame-rate Tracking System) for low frame-rate videos that utilizes existing traffic cameras and can be readily deployed at scale to provide immediate benefits for cities, pedestrians, and drivers. *LFTSys* is adaptable to different environments and camera perspectives to be flexible to different parts of the city. Finally, *LFTSys* can be deployed immediately in cities with low frame-rate camera networks such as New York City, Los Angeles, and Austin. The contributions of this work are as follows:

(1) We present the architecture, design and implementation of a real-time video analytics pipeline for vehicle

(a) Two consecutive frames from two low frame-rate cameras, demonstrating the high displacements of vehicles.

(b) Different environments, perspectives, and lighting conditions observed by the cameras.

Figure 1: Key challenges addressed by *LFTSys*.

tracking and estimating traffic flow in low frame-rate cameras.

(2) We present novel algorithms that leverage physical knowledge of the environment and are suited for real-time computation and is robust to occlusion.

(3) We evaluate *LFTSys* on New York City's publicly-available network of low frame-rate traffic cameras, and demonstrate the flexibility and adaptability of *LFTSys* to different environments and camera perspectives.

2 RELATED WORKS

There are a number of possible solutions for estimating vehicle flow. One type of solution is in-road sensors, or sensors physically implanted in the road. Examples of in-road sensors include inductive-loop detectors (ILD), which measures changes in inductance when vehicles pass over the loop; cement-based piezoelectric sensors [12]; and various magnetic sensors [4]. Studies have shown detection accuracy up to 92% [10, 20] for ILDs and 99% for magnetic sensors [4]; however, the high cost of installation and maintenance [15] makes this solution difficult to deploy at scale in dense city streets.

There are a number of studies for estimating vehicle flow via vehicle detection and tracking. Common methods first subtract the background image, segment the vehicles in the image, and use a combination of features to track the vehicles [1, 7, 16, 17]. These studies have shown high accuracy (90% to 96%) and can be implemented in real-time; however, the tracking algorithms such as Kalman filtering or distance of features have substantial difficulties with large object displacements common in low frame-rate cameras.

Other methods utilize a detection line or region to count entering and exiting vehicles [11, 13, 18]. Similarly, due to high displacements in low frame-rate cameras, vehicles may occasionally skip the detection line or region entirely, which impacts the accuracy of these systems [22]. There are also studies demonstrating high vehicle counting accuracy in different environmental conditions [9, 27]. However, these studies requires high frame-rate video, and are not implemented in real-time.

Finally, there have been a few recent studies for vehicle counting in low frame-rate cameras [23, 25, 26]. In [25, 26], the authors focus on traffic density using fully connected networks. These networks are trained to identify the density of vehicles in the frame rather than individually identify vehicles. This limits the system from estimating vehicle flow, as vehicles cannot be tracked between consecutive frames. In contrast, [23] estimates vehicle flow in low frame-rate cameras, but is only tested on a single camera and is not implemented as a real-time system.

3 SYSTEM REQUIREMENTS AND KEY IDEAS

There are two key requirements addressed in this work: real-timeliness and accuracy under camera constraints. Firstly, the system should meet real-time constraints, meaning the processing runtime of a single frame should be completed before the arrival of the next frame. Real-timeliness is desirable because no additional memory is required to store video for analysis at a later time. Furthermore, smart cities applications can utilize real-time data to provide immediate information to end users.

To meet real-time constraints, the algorithms used in the computer vision pipeline should reliably complete within 1 second. A real-time system requires algorithms for both detecting vehicles, as well as matching vehicles within frames. While there are many existing options for performing these

tasks, the runtimes of these algorithms are critically important. The key idea is to incorporate algorithms that have low runtimes and are computationally efficient. Algorithms used in this work are designed with the aim of low computational complexity (to accommodate both dense and light traffic), and shallow neural network architectures are given higher priority.

Secondly, the system should provide an accurate count of passing vehicles within frame-rate and resolution limitations. New York City's traffic cameras provide feeds with low resolution and low frame-rate, which is disadvantageous for accuracy. The camera limitations provide two obstacles: the low resolution decreases object detection accuracy, and the low frame-rate disallows the usage of traditional optical flow methods.

To address resolution and perspective, we examined several object detection networks and concluded that many modern networks are capable of detecting low resolution vehicles with better than $80-90\%$ accuracy. However, a simple count of the vehicles in each frame is not sufficient to determine vehicle flow due to double counting in consecutive frames. Since existing computer vision algorithms are unable to accurately track objects in low frame-rate videos, we utilize features extracted from the vehicle image and a matching algorithm to avoid double counting.

4 SYSTEM DESIGN

As discussed in Section 3, *LFTSys* must meet specific design requirements: (i) *Real-time*: computation on a single frame must be completed before the arrival of the next frame. (ii) *Accuracy*: the vehicle flow estimate should be highly accurate despite low resolution and low frame-rate.

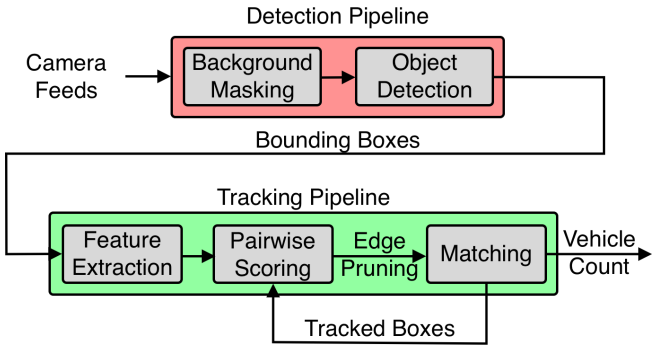


Figure 2: Architecture of the vehicle flow estimation system.

Architecture

The system architecture is shown in Figure 2. The architecture consists of a detection pipeline and a tracking pipeline. Initially, single frames are captured from an image stream, which refreshes at a frequency of 0.3 to 1 Hz. The image is fed into the detection pipeline, described in Section 4, which produces a list of bounding boxes representing detected vehicles. The list of candidate vehicles are compared with the candidates from the previous frame(s), as in Section 4. Using a combination of features and geometric constraints, candidate vehicles are matched between the two frames; any vehicles that remain unmatched are added to the global vehicle count.

Detection

Model Comparisons. To meet the real-time design requirements, we build on top of pre-trained neural networks that are fast and relatively accurate. In particular, we evaluate two different state-of-the-art networks: SSD-Mobilenet and Mask R-CNN for accuracy and computation time. These networks represent two state-of-the-art detection and segmentation algorithms. In deciding which network to use, we evaluated their runtime and accuracy performance on image stills taken from various traffic cameras. Due to the faster runtime, we chose SSD-Mobilenet as the basis for our detection pipeline.

Pipeline. The detection pipeline is composed of two parts: background masking and object detection as shown in Figure 3. Initially, the background of the images of the video stream are masked as in [26]. This ensures that low resolution patterns in the background will not be falsely detected as vehicles.

The masked image is then fed to an object detection network. To increase accuracy, we utilize training data from two sources: the CityCam dataset [26], and a custom hand labeled dataset comprising of two thousand images and approximately five thousand vehicles. We use transfer learning to tune the pre-trained SSD-Mobilenet model to better recognize vehicles at low resolution. Transfer learning is achieved by freezing all layers except for the final layer, and retraining the neural network.

Tracking

Vehicle tracking between frames is the most challenging part of this work due to the low frame-rate. However, tracking is necessary to avoid double counting vehicles between frames. A block diagram of the tracking pipeline is shown in Figure 4. We consider vehicles detected in the current frame, as well as in a fixed number of previous frames. The detection pipeline produces a new set of bounding boxes B for each incoming frame, and the previous set of bounding boxes B' is kept as a reference for matching.

Feature Extraction Feature extraction is critical for matching vehicles between consecutive frames. Certain features may have advantages in certain types of environments; thus, multiple types of features are extracted from each vehicle to maximize the probability of correct matching. We selected

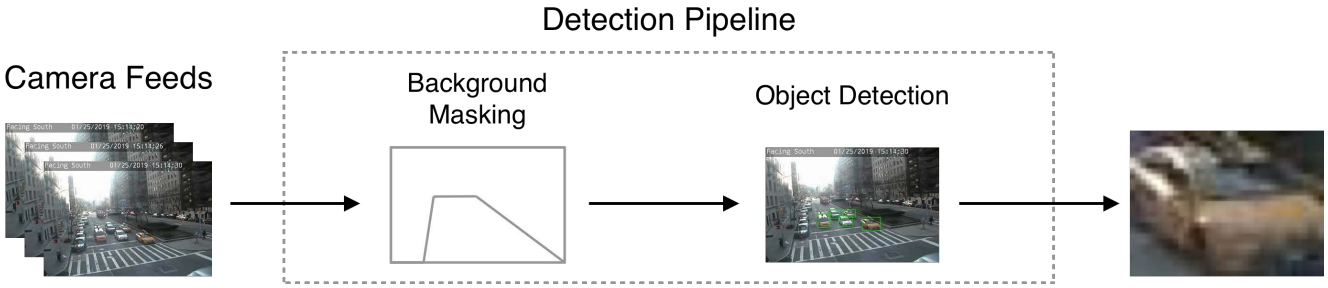


Figure 3: Computer vision pipeline for detecting vehicles. A) the background is masked from the incoming video stream image, and B) an object detection network extracts the candidate bounding boxes.

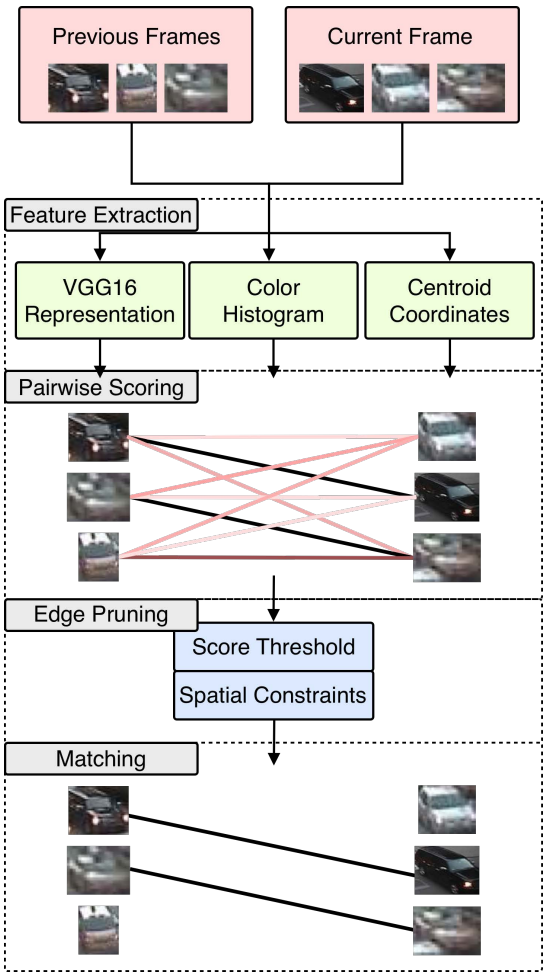


Figure 4: Pipeline for tracking vehicles with large displacement. Features are extracted from two candidate vehicles from consecutive frames.

two types of features which are complimentary, and are robust to high displacement in certain situations: VGG16's conv3_3 layer output, and color histogram. The first type of feature is extracted from VGG16's conv3_3 layer, as shown in Figure 5. In [8], the authors propose that in the outputs of VGG16's higher layers, image content and overall spatial structure are preserved, but localized features such as color, texture and exact shape are not. After testing outputs of different layers of VGG16, we chose to use the output of layer conv3_3 as the prominent features in our tracking pipeline. Figure 5 shows the original VGG16 architecture, and the conv3_3 layer which outputs the vehicle representations. Using the output of this layer has the added benefit of reducing the computational runtime, as inputs do not have to pass through the entire VGG16 architecture.

The second type of feature, color histogram (CH), was chosen to complement the VGG16 representation. This feature is extracted by computing a histogram of the number of pixels in the vehicle image of a certain color. Since bounding boxes may vary in size, the histogram is also normalized.

For each bounding box $b \in B$, a VGG16 representation and color histogram are computed. The newly computed features form a new set U. The VGG16 representations and color histograms of the previous frame bounding boxes are stored in V for later matching.

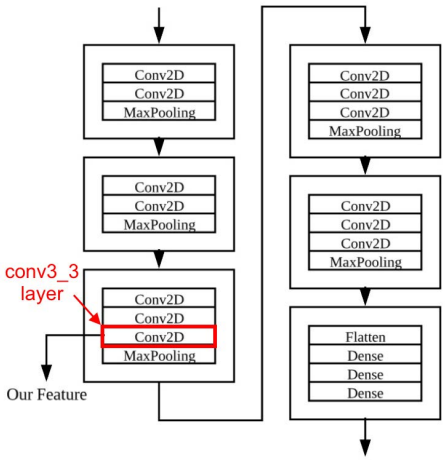


Figure 5: Original VGG16 architecture and an indication of the conv3_3 layer used for producing vehicle feature representations.

Pairwise Scoring Between two consecutive frames, we utilize pairwise scoring to help determine which candidate pairs are likely to be the same vehicle. From the sets U and V, we compute two correlation scores for each pair in the Cartesian Product $U \times V$, using both the VGG16 representations and the color histograms. The final matching score is computed as a linear combination of the VGG16 and color histogram scores. The α parameter allows for flexibility in prioritizing the features that provide higher matching accuracy.

$$s_{ij} = (1 - \alpha)VGG16_{ij} + \alpha CH_{ij}$$

Once the final scores have been calculated for each pair of features, a bipartite graph is constructed. The bipartite graph consists of two sets, U and V representing the features of the previous and current frame's bounding boxes, respectively. Edges between elements of the two sets represent the final matching score.

Edge Pruning To reduce the chances of mistakenly matching two different vehicles, we first prune the edges of the bipartite graph using two criteria. The first criteria is a score threshold. Intuitively, two vehicles that are sufficiently dissimilar should have a low score; thus, any edge in the bipartite graph with a score below a certain threshold is pruned. The setting of this threshold is discussed in Section 5.

The second criteria is a spatial constraint. In most cases, vehicles must travel in the direction of traffic flow. Thus, edges between bounding boxes which violate spatial constraints are pruned. The spatial constraints are manually specified for each environment, and is verified by checking the change in the centroids of the bounding boxes. In Figure 6, the green marked vehicle in the first frame has a high matching score with the red and green boxes in the second frame. Due to spatial constraints (centroid advances towards the camera), the vehicle is matched correctly.

Figure 6: Example of a spatial constraint. The red dotted line indicates a horizontal spatial constraint.

Matching

Once the bipartite graph has been constructed and pruned, a matching must be found to best correlate the vehicles. In graph theory, a *matching* is a set of edges in a graph without common vertices. In this application, we seek a matching such that each vehicle in frame X is matched with at most one other vehicle in frame $X + 1$, and vice versa; further, we aim to maximize the scores between the matched vehicles.

This problem is commonly referred to as *maximum weighted bipartite matching*, as well as the *assignment problem*. The problem is formulated as follows. We have two sets of equal size, U and V, and a weight function $S : U \times V$, which is the matching score in our application. The goal is to maximize the total weight, while ensuring that the matching forms a bijection between U and V. This results in the following linear program which can efficiently be solved in polynomial time:

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in U\times V} s_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{i\in U} x_{ij} \leq 1, \ \forall j \in V \\ & \sum_{j\in V} x_{ij} \leq 1, \ \forall i \in U \\ & 0 \leq x_{ij} \leq 1, \ \forall i, j \in U, V \end{aligned}$$

Where s_{ij} is the final pairwise score between vehicle representations $i \in U$ and $j \in V$, and x_{ij} is the decision of match (1) or no match (0). Since the sets U and V are not necessarily the same size, extra "dummy" nodes are added to the smaller set to make the sets equal. Further, any missing edges are given weight 0 to ensure the bipartite graph is *complete*.

One of the challenges facing matching algorithms is occlusion. Traffic cameras in dense inner-city environments are especially susceptible to occlusion; due to the low angle of many of the cameras, vehicles can often become partially or completely occluded by larger vehicles.

In many scenarios, objects may no longer be detected due to occlusion or due to the object leaving the frame. To address this issue, we introduce **memory**; any unmatched vehicle representations remain in memory (set U) for future matching. Since more recent vehicle representations should be prioritized in matching over representations further in the past, we apply a discount γ to the pairwise scores for every frame that has passed. When the discount causes the maximum score to fall below the score threshold in edge pruning, the representation is removed from the set U.

One final consideration for the discount is the amount of change in the image. When the image is not significantly changing, such as due to a stop light, the past representations should not be discounted as heavily as when the image is significantly changing. Thus, we set the discount equal to a scaling factor multiplied by the correlation between frames X and $X + 1$; this ensures that the discount will be sensitive to the change in the image, while allowing for past vehicle representations to potentially be matched again.

An example is shown in Figure 7 of a taxi marked with a red box. A-B) The taxi is initially tracked; C) the vehicle is occluded by surrounding vehicles, and the vehicle's feature representation is kept in memory; D) after a number of frames, the vehicle is no longer occluded, and is matched with the feature representation in memory.

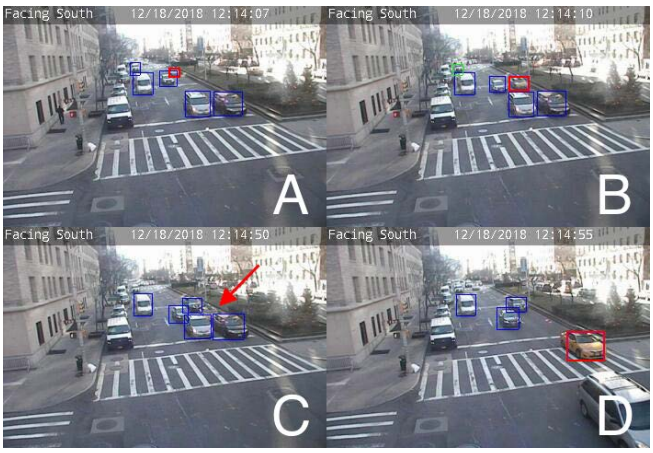


Figure 7: Example tracking over multiple frames with occlusion.

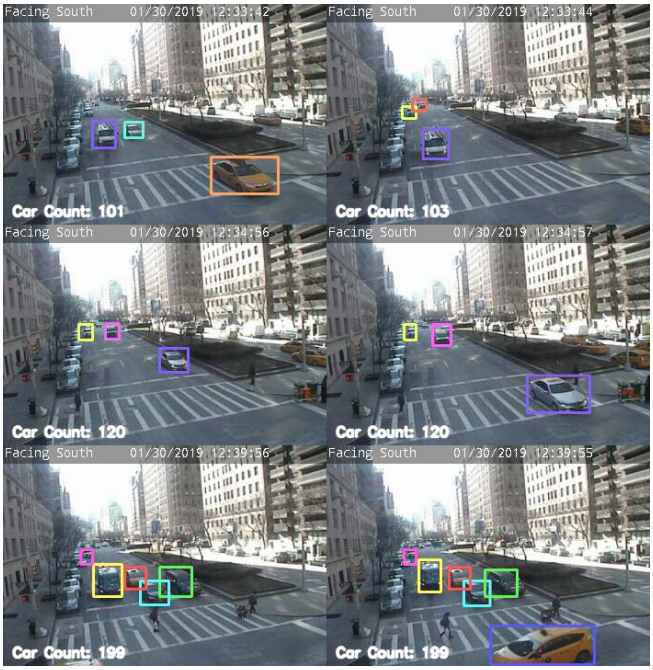


Figure 8: Vehicle matching in successive frames. Matched vehicles are annotated with the same colored boxes.

5 IMPLEMENTATION AND EVALUATION

Setup and Data Sources

The primary source of data used in the implementation and evaluation of *LFTSys* is the New York City's Department of Transportation traffic cameras, which is publicly available at [19]. The camera network consists of 752 closed circuit cameras which transmit images at a regular frequency (0.3-1 Hz). We captured image sequences from 20 representative cameras throughout the city to vary the diversity of environmental conditions. An example of *LFTSys* running on an image sequence is shown in Figure 8.

Detection

We compared two types of object detectors (SSD-Mobilenet [14] and Mask-RCNN [6]) for accuracy and computational runtime, as discussed in Section 4. Their statistics are shown in Table 1.

Model	Runtime μ	Runtime σ	Precision	Recall
SSD-Mobilenet	0.057 s	0.0048 s	0.976	0.892
Mask-RCNN	1.808 s	0.0472 s	0.862	0.803

Table 1: Comparisons of SSD-Mobilenet and Mask-RCNN.

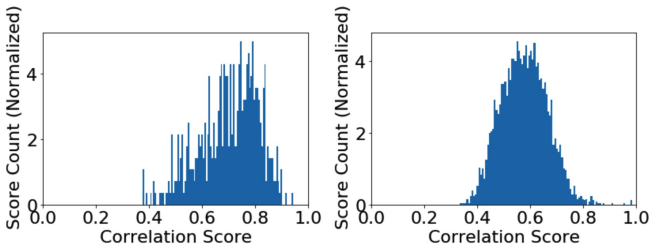
To satisfy the real-time requirement in our implementation, we utilize SSD-Mobilenet with the faster computation time in the detection pipeline. This part the pipeline can be substituted with another detection network if the runtime meets the real-time constraints.

Tracking

To evaluate our tracking methodology, we first evaluate the matching potential of the VGG16 representations alone. We proceed to evaluate the VGG16 representations and color histogram features together. The image sequences described in Section 5 are used for testing the matching accuracy.

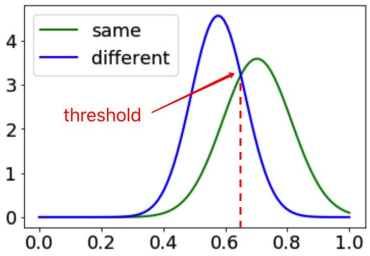
Feature Extraction. As mentioned in Section 4, we use the first three convolutional blocks of VGG16 as our feature extraction network. However, instead of using original VGG16's input size $224 \times 224 \times 3$, we re-size all the detected cars to $48 \times 48 \times 3$. This is the closest size to the average size of detected vehicles in a typical image, and is a feasible size for the three blocks of VGG16. With the input size of $48 \times 48 \times 3$, the extracted features are flattened to produce a 9216 length vector representation for each vehicle.

Correlation. Initially, we evaluated the potential of correlation for matching vehicles. We constructed a dataset of vehicle bounding boxes from the detection pipeline, and used the vector representations to test correlation. As shown in Figure 9, the correlation scores of same vehicles and different vehicles are plotted as histograms in a) and b). The



(a) Histogram of correlation scores of pairs of same vehicle images.

(b) Histogram of correlation scores of pairs of different vehicle images.



(c) Distribution approximations and threshold computation.

Figure 9: Method for computing correlation threshold from a dataset of vehicle images.

distributions of the two histograms produce a best threshold for the separation of same vehicles and different vehicles, as shown in c). Using the correlation method, we achieved an 89% matching accuracy on the image sequences.

System Evaluation

To demonstrate the advantages of *LFTSys*, we evaluate against state-of-the-art methods and against ground truth. To evaluate counting accuracy, we compiled images in five minute sequences from 20 different cameras. Ground truth is obtained manually; each image is compared with the previous and annotated with the number of new vehicles. The image sequences are obtained at 1 Hz on average, and contain about 100 unique vehicles for each sequence. Each method is tested on the sequence and compared to ground truth to determine accuracy.

We compare against three representative state-of-the-art techniques: 1) Mosse Tracking [2], a type of correlation filtering tracker; 2) Deepgaze [21], using particle filters; and 3) optical flow, which we implement using OpenCV [3].

Total Accuracy. For ground truth, the vehicle count for each frame is manually labeled. In Table 2, we show the mean absolute error (MAE) and root mean squared error (RMSE) of each method on the test videos. *LFTSys* runs with the lowest MAE and RMSE. Table 2 displays the overall improvement of *LFTSys* over the other state-of-the-art algorithms.

Algorithm	MAE	RMSE
LFTSys	**5.80**	**2.53**
Mosse Tracking	43.69	73.90
Deepgaze	70.25	114.89
Optical Flow	41.05	75.41

Table 2: Total system evaluation against state-of-the-art algorithms for various cameras.

Runtime. We timed each part of the system to test whether *LFTSys* can be implemented in real-time. Experiments were run on a 3.2 GHz Intel Core i5. Table 3 shows the mean, minimum and maximum runtime of each component and for the whole system on the test video sequences. The main runtime bottleneck is the SSD-Mobilenet detector; in the worst case, *LFTSys* can only run in real-time on image feeds with less than 0.7 Hz frequency. However, we conclude that the system is able to run in real-time for greater than 99% of the frames. Further, the system can achieve true real-timeliness with a faster processor or GPU.

Component	Mean	Min	Max	Avg Runs
SSD-Mobilenet	51.3 ms	42.3 ms	**1.46 s**	1
Feature Extraction	78.9 ms	8.2 ms	166.7 ms	1
Pairwise Scoring	1.66 ms	1.41 ms	5.87 ms	16
Edge Pruning	11.6 μs	10.3 ns	701 μs	32
Matching	71.7 μs	5.0 μs	3.7 ms	1
Total Runtime	51.8 ms	42.6 ms	**1.519 s**	1

Table 3: Mean, minimum, and maximum runtime of each component in the system and the total system.

Different environments. *LFTSys* is also tested on various cameras throughout New York City to demonstrate adaptability to different environments, lighting conditions, and perspectives. As shown in Figure 10, *LFTSys* is able to maintain accuracy in different environments and perspectives. Table 4 presents the mean absolute error and root mean squared error for video feeds taken from urban, rural and highway environments. Although Deepgaze performs better in the rural videos, *LFTSys* has comparable performance, and has significantly better performance in urban and highway environments than the other algorithms.

6 CONCLUSION

In this work, we present a real-time system using computer vision and deep learning pipeline to accurately track vehicles and estimate traffic flow using existing traffic cameras with low frame-rate and low resolution videos. We adopt a two part system consisting of a detection pipeline and a tracking pipeline. Detection is achieved using SSD-Mobilenet to detect bounding boxes; the bounding boxes are then tracked using features extracted from VGG16 and color histograms

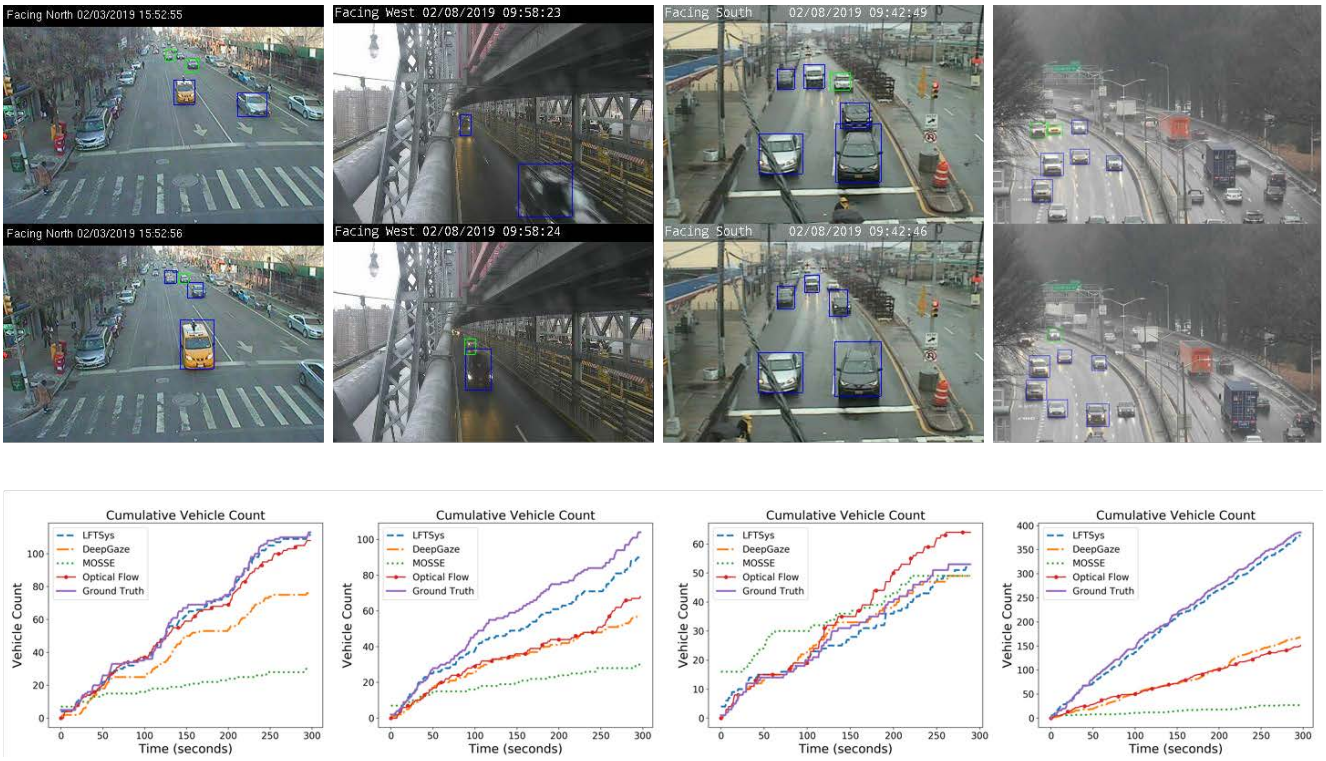


Figure 10: Vehicle matching in urban, rural, and highway environmental conditions.

Algorithm	Urban		Rural		Highway	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
LFTSys	**5.36**	**6.21**	2.77	3.159	**9.65**	**10.46**
Deepgaze	21.4	24.46	**1.7**	**2.25**	129.12	143.16
MOSSE	39.8	46.7	6.99	8.31	192.77	219.13
Optical Flow	13.4	18.1	5.37	6.83	131.16	148.09

Table 4: Total system evaluation against state-of-the-art algorithms for various cameras.

as well as an efficient linear program matching algorithm. We demonstrate the system's high vehicle counting accuracy over other existing algorithms even in low frame-rate and low resolution image feeds, and the system's adaptability to multiple types of environments.

ACKNOWLEDGMENTS

This research was partially supported by the National Science Foundation under Grant Numbers CNS-1704899 and CNS-1815274. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed implied, of Columbia University, NSF, or the U.S. Government or any of its agencies.

REFERENCES

[1] Prem Kumar Bhaskar and Suet-Peng Yong. 2014. Image processing based vehicle detection and tracking method. In *Computer and Information Sciences (ICCOINS), 2014 International Conference on*. IEEE, 1–5.

[2] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. 2010. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2544–2550.

[3] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[4] Sing Yiu Cheung, Sinem Coleri, Baris Dundar, Sumitra Ganesh, Chin-Woo Tan, and Pravin Varaiya. 2005. Traffic measurement and vehicle classification with single magnetic sensor. *Transportation Research Record* 1917, 1 (2005), 173–181.

[5] Daniel de Godoy, Bashima Islam, Stephen Xia, Md Tamzeed Islam, Rishikanth Chandrasekaran, Yen-Chun Chen, Shahriar Nirjon, Peter R Kinget, and Xiaofan Jiang. 2018. Paws: A wearable acoustic system for pedestrian safety. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 237–248.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2980–2988.

[7] Deng-Yuan Huang, Chao-Ho Chen, Wu-Chih Hu, Shu-Chung Yi, Yu-Feng Lin, et al. 2012. Feature-based vehicle flow analysis and measurement for a real-time traffic surveillance system. *Journal of Information Hiding and Multimedia Signal Processing* 3, 3 (2012), 279–294.

[8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*. Springer, 694–711.

[9] Shiva Kamkar and Reza Safabakhsh. 2016. Vehicle detection, counting and classification in various conditions. *IET Intelligent Transport Systems* 10, 6 (2016), 406–413.

[10] Yong-Kul Ki and Doo-Kwon Baik. 2006. Vehicle-classification algorithm for single-loop detectors using neural networks. *IEEE Transactions on Vehicular Technology* 55, 6 (2006), 1704–1711.

[11] Manchun Lei, Damien Lefloch, Pierre Gouton, and Kadder Madani. 2008. A video-based real-time vehicle counting system using adaptive background method. In *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*. IEEE, 523–528.

[12] Zhong-Xian Li, Xiao-Ming Yang, and Zongjin Li. 2006. Application of cement-based piezoelectric sensors for monitoring traffic flows. *Journal of transportation engineering* 132, 7 (2006), 565–573.

[13] Fei Liu, Zhiyuan Zeng, and Rong Jiang. 2017. A video-based real-time adaptive vehicle-counting system for urban roads. *PloS one* 12, 11 (2017), e0186098.

[14] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[15] Luz Elena Y Mimbela and Lawrence A Klein. 2000. Summary of vehicle detection and surveillance technologies used in intelligent transportation systems. (2000).

[16] HS Mohana, M Ashwathakumar, and G Shivakumar. 2009. Vehicle detection and counting by using real time traffic flux through differential technique and performance evaluation. In *Advanced Computer Control, 2009. ICACC'09. International Conference on*. IEEE, 791–795.

[17] Brendan Morris and Mohan Trivedi. 2007. Real-time video based highway traffic measurement and performance monitoring. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*. IEEE, 59–64.

[18] Zakaria Moutakki, Imad Mohamed Ouloul, Karim Afdel, and Abdellah Amghar. 2018. Real-Time System Based on Feature Extraction for Vehicle Detection and Classification. *Transport and Telecommunication Journal* 19, 2 (2018), 93–102.

[19] New York City Department of Transportation. 2019. Real Time Traffic Information. https://webcams.nyctmc.org/. Accessed: 2019-02-11.

[20] Herivelton A Oliveira, Fabio R Barbosa, Otacilio M Almeida, and Arthur PS Braga. 2010. A vehicle classification based on inductive loop detectors using artificial neural networks. In *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference on*. IEEE, 1–6.

[21] Massimiliano Patacchiola and Angelo Cangelosi. 2017. Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods. *Pattern Recognition* 71 (2017), 132–143.

[22] Chomtip Pornpanomchai, Thitinut Liamsanguan, and Vissakorn Vannakosit. 2008. Vehicle detection and counting from a video frame. In *Wavelet Analysis and Pattern Recognition, 2008. ICWAPR'08. International Conference on*, Vol. 1. IEEE, 356–361.

[23] Evgeny Toropov, Liangyan Gui, Shanghang Zhang, Satwik Kottur, and José MF Moura. 2015. Traffic flow from a low frame rate city camera. In *Image Processing (ICIP), 2015 IEEE International Conference on*. IEEE, 3802–3806.

[24] Stephen Xia, Daniel de Godoy, Bashima Islam, Md Tamzeed Islam, Shahriar Nirjon, Peter R Kinget, and Xiaofan Jiang. 2019. Improving Pedestrian Safety in Cities using Intelligent Wearable Systems. *IEEE Internet of Things Journal* (2019).

[25] Shanghang Zhang, Guanhang Wu, Joao P Costeira, and José MF Moura. 2017. Fcn-rlstm: Deep spatio-temporal neural networks for vehicle counting in city cameras. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 3687–3696.

[26] Shanghang Zhang, Guanhang Wu, Joao P Costeira, and José MF Moura. 2017. Understanding traffic density from large-scale web camera data. *arXiv preprint arXiv:1703.05868* (2017).

[27] Yunsheng Zhang, Chihang Zhao, and Qiuge Zhang. 2016. Counting vehicles in urban traffic scenes using foreground time-spatial images. *IET Intelligent Transport Systems* 11, 2 (2016), 61–67.