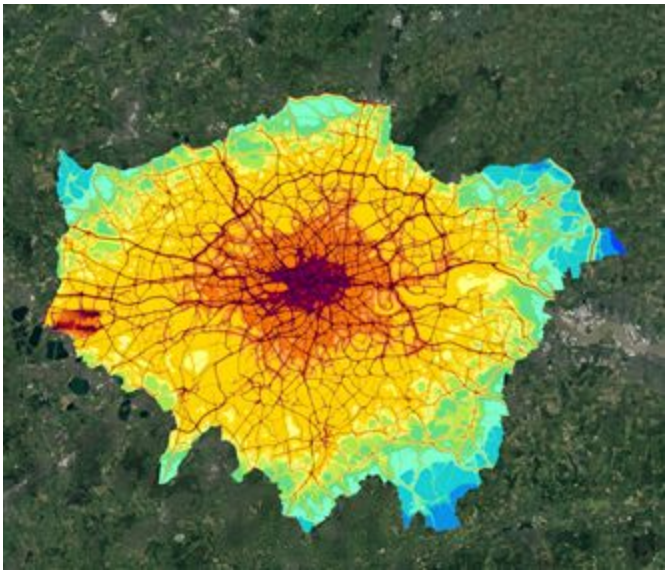
Quantifying traffic dynamics to better estimate and reduce air pollution exposure in London



Technical Report

Sam Blakeman, Jack Hensley, Oluwafunmilola Kesa, Caroline Wang

Project manager: Sam Short
Technical mentor: Maren Eckhoff

Data Science For Social Good

Summer Fellowship

Imperial College Business School | Gandhi Centre for Inclusive Innovation

THE UNIVERSITY OF CHICAGO

Abstract

Traffic is largely responsible for the air pollution that causes harm to human health in London. Our partners want to understand how traffic contributes to air quality and how proper traffic interventions can reduce exposure to pollution, but the data they use to construct emissions models is not at the necessary scope or granularity, inducing severe underestimates in pollution concentrations. The goal of this project was to uncover the information embedded in the 900+ traffic cameras dispersed across London to provide traffic statistics on vehicle counts and stop/start events near real-time and at the junction-level to fill this important gap.

Starting in June 2019, video data -- in the form of 10 second clips updated every 4 minutes -- was collected from Transport for London's API and stored on an Amazon Web Services S3 bucket for all cameras. Using techniques in the computer vision field, we processed individual video clips by detecting and tracking vehicles in the camera view. For each video clip, for each vehicle type (car, bus, motorbike, and truck), we reported counts and stop/start events. Several models were developed that combined a variety of object detection and tracking algorithms. For each model, we evaluated its performance on 42 hand-labeled videos according to processing speed and video-level accuracy. The models with the best performance according to speed and accuracy using either a CPU or GPU are described in table below.

Detection model	Detection confidence threshold	Detection frequency (frames)	Tracking algorithm	Stop/start IOU threshold	Bias (vehicles)	Processing speed (sec)
Yolov3-tiny (CPU)	0.2	4	CSRT	0.75	6.98	2.92
Yolov3-traffic (GPU)	0.5	4	MOSSE	0.75	3.78	0.36

We have provided three deliverables to meet our project partners' needs: an open-source library that contains the scope of the work of this project, a dataset containing traffic statistics in the Ultra-Low Emission Zone (ULEZ), and a Web application that allows partners to easily view historical and live data.

I. Problem Background and Social Impact

Every year, thousands of Londoners die prematurely because they are exposed to air that exceeds pollution thresholds. It is well understood that traffic is the major source of pollutants from emissions in London, but its contribution on fine spatiotemporal scales remains ill-defined.

Our partners work in tandem to understand the complex relationships between traffic and air quality to help policymakers develop appropriate strategies to reduce air pollution sourced from vehicle emissions. They include:

- The Transport and Environmental Laboratory (Department of Civil and Environmental Engineering, Imperial College London) has a mission to advance the understanding of the impact of transport on the environment, using a range of measurement and modelling tools.

- Transport for London (TFL) is an integrated transport authority under the Greater London Authority, which runs the day-to-day operation of London's public transport network and manages London's main roads. TFL leads various air quality initiatives including the Ultra Low Emission Zone in Central London.

- City of London Corporation (CLC) is the governing body of the City of London (also known as the Square Mile), one of the 33 administrative areas into which London is divided. Improving air quality in the Square Mile is one of the major issues the CLC will be focusing on over the next few years.

- The Alan Turing Institute is the national institute for data science and artificial intelligence. Its mission is to advance world-class research in data science and artificial intelligence, train leaders of the future, and lead the public conversation.

With the help of our partners, policymakers in London have made significant efforts towards tackling pollution by introducing legislation that attempts to specifically curtail traffic emissions. A recent example is the introduction of the Ultra-Low Emissions Zone (ULEZ), which requires drivers that pass through the city center to meet emissions requirements or pay a fee. While it has been reported that the ULEZ reduced the number of 'worst polluting cars' in Central London[1], there has been no quantification of how it has affected traffic levels inside or outside of the ULEZ and whether emissions and air quality have improved or worsened. There is an acute need to improve the capability to understand how traffic disruptions and policies affect traffic congestion--and in turn, how this affects vehicle emissions and air pollution.

In conjunction with direct air quality measurements, air quality modeling enables air quality forecasting and prediction of the effects of future traffic initiatives on air quality. Air quality models require emissions inputs that are near the granularity of the models themselves, e.g., to know the air quality at the road-level every hour, emissions should also be estimated at the road level every hour. However, the

[1] https://www.theguardian.com/uk-news/2019/may/16/ulez-cuts-number-of-worst-polluting-cars-in-central-london

traffic data available is only reported as vehicle counts at the regional level (i.e., London-wide) on an annual scale. Stop-start events, i.e., when a vehicle slows to a stop or accelerates from a stop, are not considered, even though these events contribute most to air pollution, especially in congested areas. When emissions statistics are broken down to the desired resolution in models, systematic underestimates in air pollution are produced.

As such, what relevant researchers and policymakers lack is an open-source library that quantifies traffic dynamics across London at high spatiotemporal resolution. Such a library would enable improved emissions model predictions, and therefore more accurate evaluation of the impact of future transport initiatives, road works and road closures, and traffic flow optimizations to reduce congestion.

At present, Transport for London (TfL) has a network of traffic cameras (JamCams) dispersed across London. The footage from these cameras is publically available on TfL's website, released as 10 second clips every four minutes. This video data therefore contains an underutilized source of traffic information with broad coverage in London.

The purpose of this project is to facilitate the extraction of meaningful traffic statistics from video data collected from the JamCams, delivered as

- An open-source library that automatically collects live traffic video data, identifies vehicles and tracks their behavior, extracts descriptive statistics, and displays the results on a Web application;
- A traffic dataset that estimates counts of vehicles and stop/start events at the junction-level, at near real-time, in the ULEZ; and
- A web application that allows users to conveniently access traffic data.

In the following sections, we describe the data, methods, results, and implications of the work conducted in this project.

II. Data Science Problem Formulation

Computer vision, i.e., extracting meaningful information out of video data, is one of the hallmark fields of data science. Our task was to use modern computer vision techniques to translate video footage into vehicle detections and ultimately obtain traffic statistics. The problem of providing finer-grained traffic information across Greater London was a collection of classification and regression problems.

More specifically, for each video we process, we were interested in answering the following questions, and formulated each as a type of data science problem:

1. What vehicle types are detected in a single frame? How many? (Single label multiclass classification)
2. What are the true vehicle counts in a single video? (Regression)
3. What are the true number of stop and start events in a single video? (Binary classification)

We required that the vehicle detections be measured at the image-level, that is for any given image, vehicle types were assigned to the appropriate objects identified in the image. However, the vehicle

counts and stop/start events were performed at the video-level because it required temporal information that could only be accessed across multiple images. We reported vehicle counts as the number of unique vehicles present over the course of a video, which meant that we need to be able to track individual vehicles throughout the video. Similarly in order to identify stop/start events we needed to be able to infer motion across a sequence of images.

The major constraints were:

- The video footage was not continuous stream but was released as 10-second clips updated every 4 minutes; this meant that there was the potential for a high degree of noise in our estimates as the 10 seconds might not have been representative of the 4 minute window. Additionally, if vehicles were stopped for long periods of time, they may have been stopped for the entire 10 second window and would be indistinguishable from parked cars.
- The resolution of the video clips (352 x 288 pixels) limited our ability to identify vehicles in the distance or to extract information such as number plate identity, which would have given a more accurate identification of vehicle type.

We made the following assumptions about each of the questions posed above, some of which were requisite given the identified constraints:

- The following vehicle labels were sufficiently detailed enough to be useful for our partners and to ultimately improve air pollution estimates: car, bus, truck, motorbike.
- Cars that were rarely stopped during the entire duration of a 10 second clip were not included in stop events.
- Parked cars were not of interest to our partners and were not included in the vehicle counts.
- An insufficient number of vehicles exited and re-entered the image, and therefore these vehicles would not influence the true vehicle counts in a vehicle.

III. Data Summary

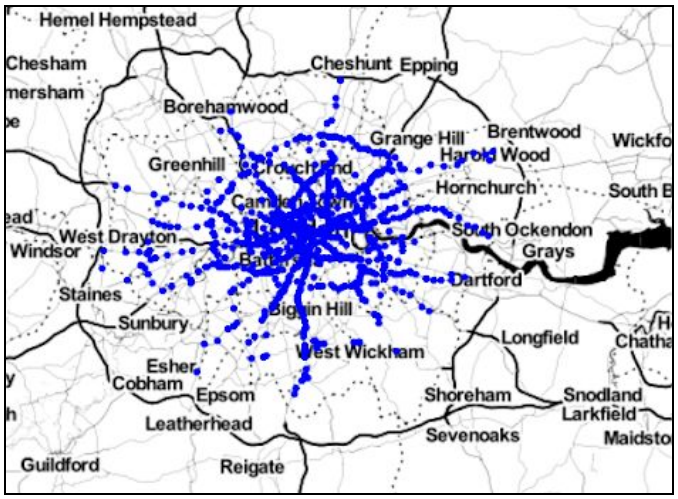


Figure 1. Locations (in blue) of traffic cameras around London.

Our data source was video clips from TfL JamCams. In total there were 911 cameras dispersed across greater London (see Figure 1). TfL posted 10 second video clips every 4 minutes for each camera. Each video clip had a resolution of 352 x 288 pixels and a frame rate of approximately 25 frames per second. At any given point in time, a subset of the cameras may have been in use by Transport for London (TfL) and as a result their video feed was blank. See Figure 2 for a screenshot of an example video clip.

Figure 2. Frame from an example video from a traffic camera.

Since June 2019, we downloaded all videos for all JamCams using the TfL API. The videos were stored in an S3 bucket on Amazon Web Services (AWS) as mp4 files. The videos were named with their upload date and camera ID. After the end date of the project, the videos of cameras in the ULEZ were downloaded to the S3 bucket and then deleted after two hours, after they were processed.

As described in the following section, chunks of videos were downloaded from S3 to the local machine before processing.

IV. Analytical Approach

In this section, the pipeline structure and the processing of the collected video data is discussed.

IV. a. Pipeline Structure

All pipelines predominantly relied upon an S3 bucket and a PostgreSQL database. In total we had three pipelines:

- Live Analysis Pipeline
- Evaluation Pipeline
- Data Collection Pipeline

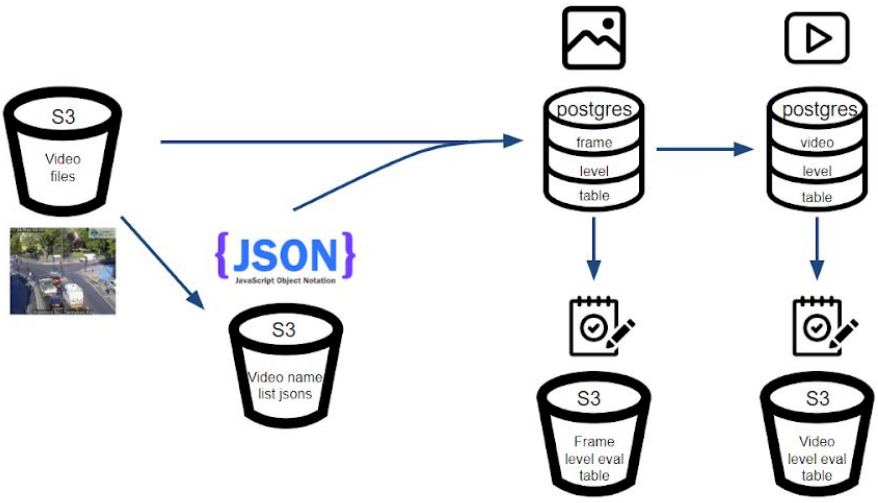


Figure 3. Schema of data flow/storage through our pipelines.

The data collection pipeline was responsible for collecting video data from the TFL API in real-time to be stored in an S3 bucket. Its steps are:

1. Using the TFL camera API (configured in parameters.xml file), download the current footage for each camera into a local file
2. In parallel with the download function, move the video files from the local directory to an s3 bucket
3. Repeat the download and upload for each camera every three minutes

The evaluation pipeline was responsible for evaluating modelling approaches using hand-annotated videos. Its steps are:

1. Create grid of parameters to assess.
2. Check for all annotations XMLs available on the S3 bucket. From XML filenames, create an annotations.json listing the corresponding video names, and upload JSON to S3 bucket.
3. Query S3 bucket for the subset of videos listed in the annotations.json. Download both the annotation XMLs and the video files locally to a temporary folder.
4. Create PostgreSQL tables to store evaluation results for all parameter sets.
5. Loop through the parameter sets; for each:
    a. Create PostgreSQL tables to store results using the current parameter set.
    b. Construct instance of traffic analyser using current parameter set.
    c. Load videos into memory as numpy arrays; pass numpy arrays to the traffic analyzer object.
    d. Traffic analyzer reports frame-level statistics.
    e. Results of the traffic analyzer are appended to a frame level table in the PostgreSQL database, created in step (a).
    f. Traffic analyzer reports video-level statistics from the frame-level dataframe created in the previous step.

g. Video-level statistics are appended to a video-level table in the PostgreSQL database, created in step (a).
h. Query video-level and frame-level results from the PostgreSQL database. Pass results to the chunk evaluator class, which compares results against the ground-truth (annotated videos), and reports evaluation statistics.
i. Append evaluation statistics for this parameter set to the PostgreSQL evaluation tables.

The live analysis pipeline was responsible for continuously processing video data in real time and updating the PostgreSQL database with the latest traffic statistics. Its steps are:

1. Every hour the videos from all the JamCams from the past hour are retrieved
2. Videos processed N videos at a time where N is a parameter set in the parameters.yml file
3. Each subset of videos is first downloaded locally to a temporary folder and then loaded into memory as numpy arrays
4. Pass numpy arrays to the traffic analyzer class
5. Traffic analyzer class reports frame-level statistics
6. Append results of the traffic analyzer to the frame level table in the PostgreSQL database
7. Traffic analyzer reports video-level results from the frame-level dataframe created in the previous step
8. Append video-level results were to the video-level table in the PostgreSQL database

IV. b. Video-level processing

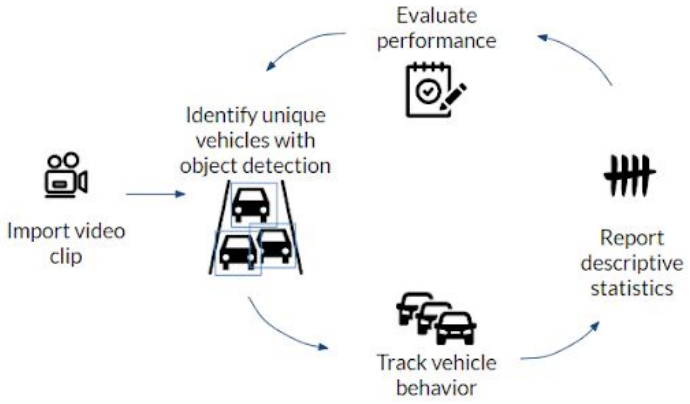


Figure 4. Schema of iterative approach to processing video clips.

Figure 4 above summarizes our approach in which we calculated vehicle behaviour descriptive statistics for a single imported video clip. Chunks of videos were selected according to the test_search.json with the list of video names, and pulled into the local machine from the S3 bucket. Each video was then processed individually within the chunk.

An imported video clip was split into frames, in which objects were detected or tracked with computer vision techniques. As discussed in the previous section, we reported statistics at both the frame and the

video level. Based on comparisons of our model's performance against hand-labelled ground truth data, we made improvements to the detection and tracking algorithms. The finalized processed approach is described below, followed by more specific details regarding the detection and tracking methods used. Points of evaluation are identified throughout the following sections, which are further discussed in Section VI, and guided our final model selection.

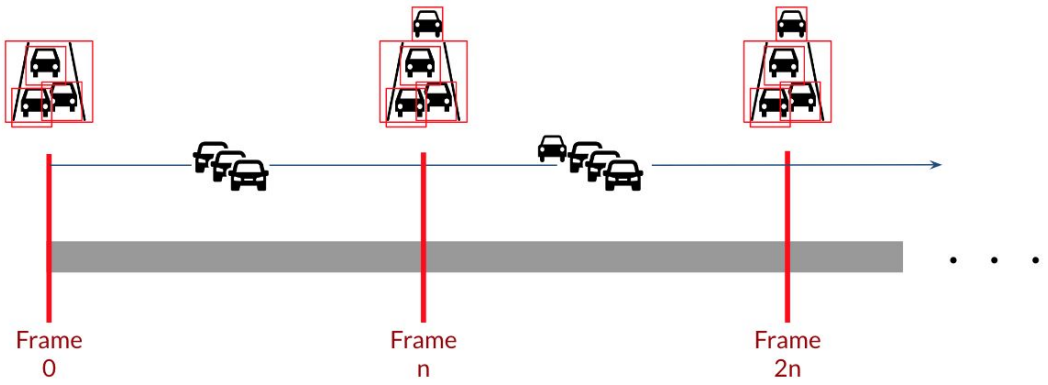


Figure 5. Cartoon of the detection (red) and tracking (gray) that occurred in the processing of a single video clip.

Figure 5 describes how we divided detection and tracking tasks among frames within a single video clip. Detection and tracking were implemented in tandem to deliver an optimal output in terms of classification accuracy and processing speed; as such, they serve specific purposes to the processing of a video. We also skipped frames to increase processing speed; the performance against interval of skipping (e.g., saving every 2nd, 3rd… frame) was evaluated.

The purpose of detection was to identify, classify, and locate all desired vehicles within the frame. Our model performed detection on the first frame, and every subsequent nth (roughly 15th) frame. Our reasoning for detecting with this frequency was that detection is computationally expensive and should not be performed every frame, however the model needed to identify new vehicles that enter the camera view, roughly every second.

The purpose of tracking was to preserve the identity of vehicles, so that we could identify detected vehicles as "new" and so that their behavior was followed across frames. After performing detection in the 0th frame, all vehicles were assigned a tracker, which followed their behavior by predicting their motion and appearance for the next n frames. In frame n, detection was repeated, and any new vehicles not already tracked would be assigned a new tracker, while tracked vehicles would maintain their existing tracker. In this way, tracking enables a realistic measure of the true counts of vehicles in a video clip. The motion of tracked vehicles was followed across frames, enabling us to identify vehicle stops and starts and when they exited the frame.

The specific methods applied to detection and tracking are discussed in the following sections.

IV. b. Detection methodology

The goal of detection was to correctly identify, classify, and locate each vehicle in a frame with a single label and a single bounding box, or region of pixels in which the vehicle occupied space within a frame, at as high a speed as possible. The computer vision community has developed many out of the box detection models that typically use neural networks to identify, label, and locate objects of interest in a frame. We chose to use You Only Look Once Version 3 (YOLOv3) as our detection model because it performed detection at high speed without sacrificing accuracy. In brief, the YOLOv3 model contained a single neural network that looked at a frame only once, and extracted features at three different scales of varying granularity. It was trained on the COCO dataset, which contains 80 objects including cars, trucks, buses, and motorcycles. The output of YOLOv3 was a list of labels, bounding boxes, and confidences in the accuracy of those bounding boxes and labels applied to each detected object. We applied post-processing to give each vehicle one detection, i.e., one label, bounding box, and confidence (Figure 6). The post-processing was a confidence threshold to delete low-confidence detections and a non-maximum suppression (NMS) threshold to delete overlapping detections. The influence of confidence and NMS threshold on model performance was evaluated.

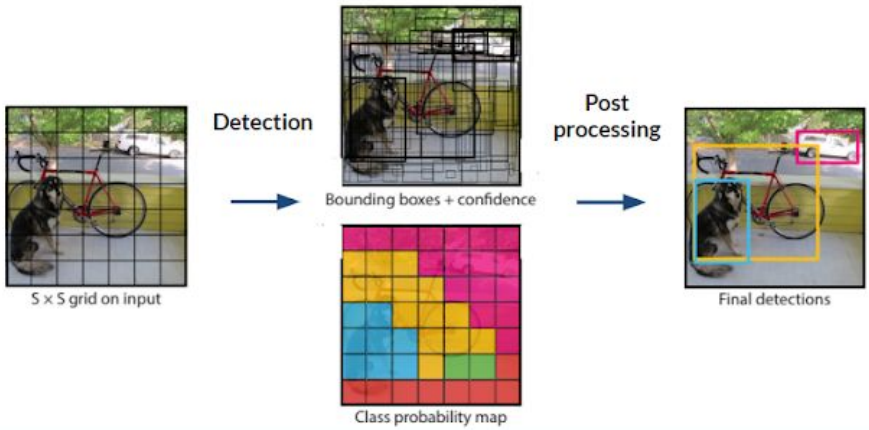


Figure 6. Procedure for performing object detections on a single image. An image was transformed into a numpy array, and a detection model proposed possible objects within the frame. We post-processed the detections with the goal of attaining only one detection per object.

YOLOv3 has two implementations, a “tiny” (YOLOv3-tiny) and a heavier weight (YOLOv3) version, that offer the user faster processing speed and lower accuracy or slower processing speed and higher accuracy, respectively. We also developed a custom detection model (YOLOv3_traffic) that used transfer learning to train the YOLOv3 network on traffic images. Rather than training the entire network, transfer learning trains only a subset of layers to preserve the bulk of the network but quickly tune the network’s ability to recognize objects on a custom dataset. To conduct transfer learning, we took the YOLOv3 network and froze all but the three penultimate layers before the three feature extractions. We passed

hand-labeled annotated JamCam images and images from the UA-DETRAC dataset[2] and output the trained model with the highest performance (i.e., with the highest mean average precision, mAP). In summary, we had three detection models to evaluate: YOLOv3-tiny, YOLOv3, and YOLOv3_traffic.

IV. c. Tracking methodology

The tracking was divided into three tasks: (1) tag newly detected vehicles with a tracker, (2) follow the motion of tracked vehicles, and (3) identify vehicle starts and stops. The methods implemented to perform tasks (1) and (3) were intersection-over-union (IOU) and CSRT tracker to perform task (2).

Intersection over union, as portrayed in Figure 7, was the area of overlap between two objects' bounding boxes over the total area that they occupy. As such, it enabled us to perform tasks that required us to know if detections occupied the same space *across* frames.

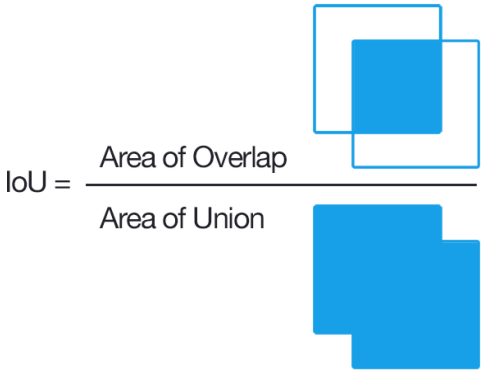


Figure 7. Cartoon depiction of intersection over union

OpenCV is a C-implemented package with a Python wrapper that performs a variety of computer vision tasks. One of particular interest to us was the Multitracker class, which assigned individual vehicles an identification tag, called a "tracker." After assignment, the tracker predicted the motion of the vehicle based on the vehicle's historical trajectory and appearance in the bounding box. The Multitracker class contained eight built-in algorithms that predicted the motion of the vehicle, each offering a particular accuracy or speed in performing tracking.

After detecting all vehicles in the nth frame with object detection, their detected bounding boxes were compared against tracked vehicles' bounding boxes from the previous frame with IOU to determine whether they were new or not. If the IOU was below a certain threshold, 0.05 in our model, then we assigned the detected vehicle a tracker. We chose 0.05 and not 0 to account for any noise in the bounding box predictions. In the 0th frame, all objects are new, so this task was not necessary.

From the 0th frame onward, we followed each tracked vehicle's motion using Multitracker. With this tracking, the model counted the total number of vehicles by class in a video clip and determine when

[2] "The UA-DETRAC Benchmark Suite - University at Albany." https://detrac-db.rit.albany.edu/. Accessed 15 Aug. 2019.

vehicles left the image. Additionally, from the 0th frame onward, we computed the IOU of each vehicle among frames to identify vehicle stops and starts. As depicted in Figure 8, vehicles that stopped would transition to an IOU ~ 1 across frames, while those that started would transition to an IOU < 1 from an IOU ~ 1. In reality, because there was noise in IOUs computed for a vehicle's bounding boxes across adjacent frames, we calculated IOUs 15 frames apart and performed a moving average to decrease noise. Via visual inspection, we identified "stops" when computed IOUs > 0.75.

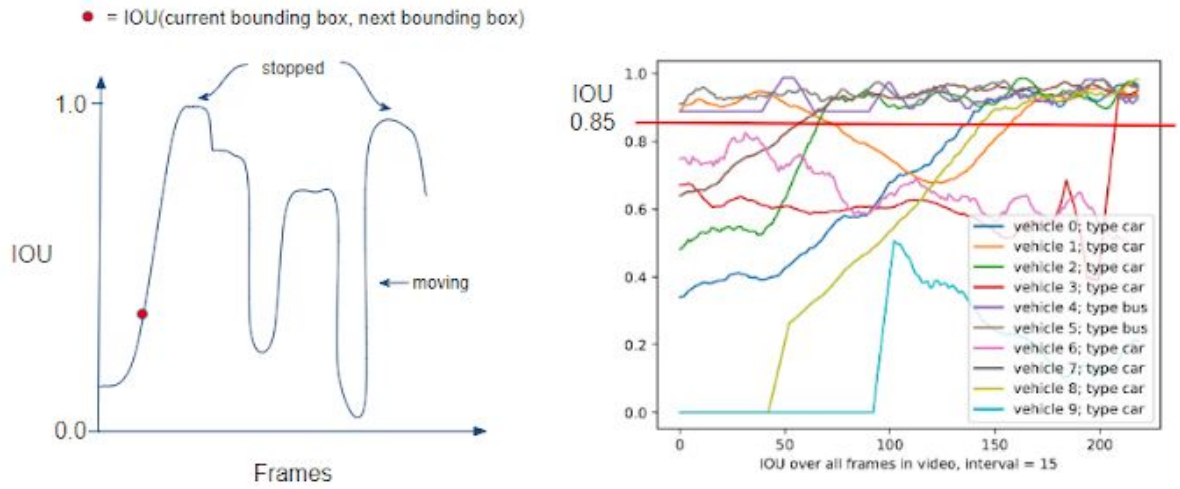


Figure 8. Idealized depiction of an IOU time series (left) for an individual vehicle across frames and actual smoothed time series (right) for all vehicles in a single video across frames. A threshold (displayed as 0.85 here) was selected for determining a vehicle to be "stopped" through the evaluation procedure.

After performing the object detection and tracking, our model produced statistics on the frame- and video-level. On the frame-level, these statistics were per vehicle detected:

- Frame number
- Vehicle type
- Bounding box coordinates

Note: frame-level statistics were used for evaluation purposes only, and were not included in the output dataset.

On the video-level, these statistics were per video processed:

- Counts of vehicle by type (e.g., car, bus)
- Counts of stop events
- Counts of start events

V. Evaluation Methodology

We evaluated our models against 42 human-annotated traffic videos, which we treated as ground truth. In this section, we describe our annotation methodology, evaluation metrics, and model selection.

V. a. Annotated data set

We used the Computer Vision Annotation Tool (CVAT) [3] to create ground-truth labels on the 42 JamCam videos we collected between June and August 2019. These videos were selected using the following criteria:

- From cameras which are well-distributed across Central London, as per Figure 9
- During days, nights, weekdays, weekends

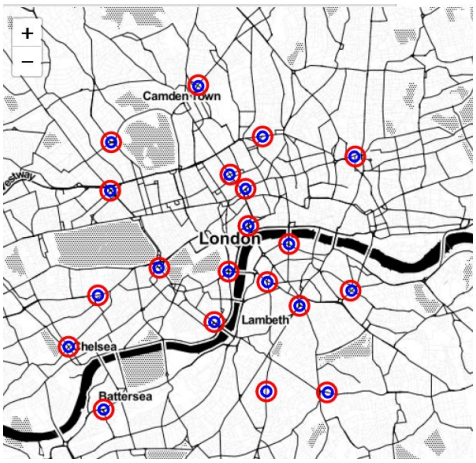


Figure 9. Location of the cameras which annotated videos were drawn from.

For each video, we labelled cars, trucks, buses, motorbikes, and vans; for each object labelled, we annotated whether it was stopped or parked. Refer to the Appendix for more details on our annotation methodology.

V. b. Evaluation Metrics

As our approach outputs both video level and frame level statistics, we chose to perform evaluation on both the video level and the frame level. As our models did not produce statistics for vans or indicate whether a vehicle was parked, these statistics are omitted from the evaluation process.

Video level evaluation is performed for all vehicle-types (cars, trucks, buses, motorbikes) and vehicle statistics (counts, stops, starts) produced by our model. For each combination of vehicle type and vehicle statistics (e.g. car counts, car stops, car starts) we computed the following summary statistics over the evaluation data set:

- Mean absolute error
- Root mean square error

As speed was a primary consideration for our project partners, we also evaluated the average runtime for each model.

[3] The CVAT tool can be found at https://github.com/opencv/cvat.

Frame level evaluation is performed for all vehicle types produced by our model. For each vehicle type and for each video, we compute the mean average precision. Mean average precision is a standard metric used by the computer vision community to evaluate the performance of object detection/object tracking algorithms. Essentially, this performance metric assesses both how well an algorithm detects existing objects as well as how accurately placed the bounding boxes around these objects are.

In the context of our project, mean average precision could be utilized to interrogate video-level model performance and diagnose issues. However, as video level statistics were the primary deliverable to our project partners, we used video level performance to select the best models.

V. c. Model grid search

The Tracking Analyser has several parameters which must be chosen by the user. To tune these parameters, we constructed a parameter grid by varying the following parameters, and evaluated Tracking Analysers constructed from these parameter sets on our annotated dataset:

- Tracking method: KCF, MOSSE, CSRT, TLD, BOOSTING, MIL
- Detection model:
    - YoloV3 with OpenCV implementation, YoloV3-tiny with OpenCV implementation,
    - YoloV3 with Tensorflow implementation, YoloV3 with transfer-learned weights and Tensorflow implementation
- Detection IOU threshold: 0.2, 0.3, 0.4, 0.5
- Detection frequency: every 4 frames, every 6 frames
- Stop start IOU threshold: 0.75, 0.80, 0.85, 0.90

V. d. Results

The model grid search was performed on 331 tensorflow models using GPU and 195 opencv models using CPU. The results of the search are shown graphically in Figure 10 below, separated by GPU and CPU, to demonstrate how each model performed in terms of processing speed and accuracy. Each point on the plot represents a different model run, using a unique set of model parameters, as described in the above section. The highest performing model in terms of speed (green) and accuracy (red) are labeled, as well as a pre-determined speed threshold to identify the maximum speed per video processed. The speed threshold is the maximum processing speed to record traffic data for at least 50 cameras per day. The evaluation is incomplete because we did not have enough time to process all models with CPU.

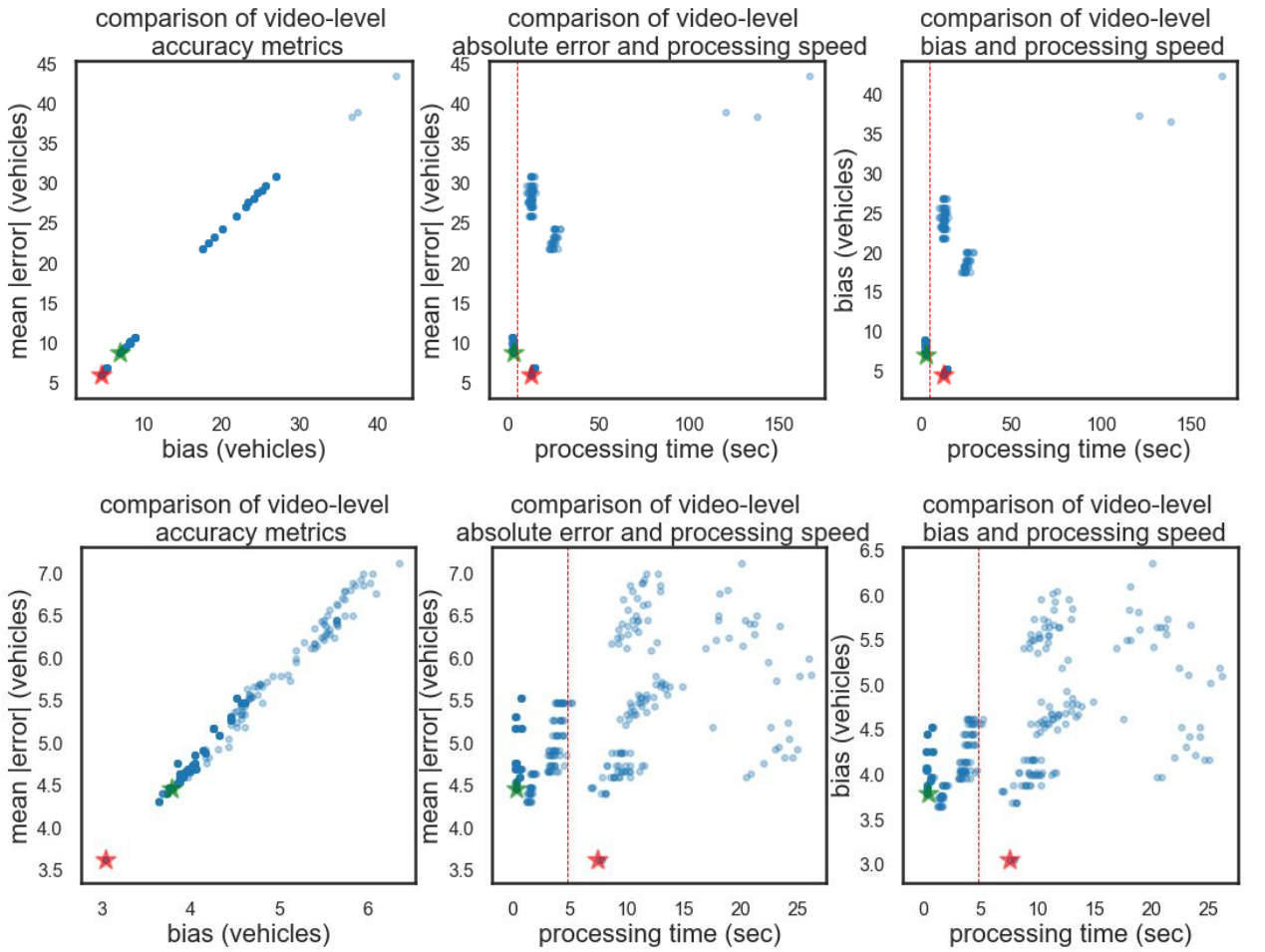


Figure 10. Performance of models in the grid search for CPU (above) and GPU (below). The model ensemble is represented by blue points, and the green star represents the highest performance model within the speed threshold, and the red star represents the highest performance model overall.

In general, a clear tradeoff between computational speed and model accuracy was not established. It is therefore possible to select a model that is fast and accurate, rather than having to select one based on a compromise between speed and accuracy. Comparing the two sets of figures, the models that were tested with GPU tended to be both faster and more accurate than the models that were tested on CPU. The best performing models on CPU are:

Detection model	Detection confidence threshold	Detection frequency (frames)	Tracking algorithm	Stop/start IOU threshold	Bias (vehicles)	Processing speed (sec)
yolov3-tiny	0.2	4	CSRT	0.75	6.98	2.92
yolov3-tiny	0.2	4	KCF	0.75	4.49	12.6

As shown in Figure 10 (lower set), there were many models that were faster than the speed threshold and at high accuracy both in terms of mean absolute error and bias. In general, the transfer learned model (yolov3-traffic) tended to perform better than the untrained model (yolov3). It is possible that the performance could be enhanced with more training. The best performance achieved is shown in the table below, achieving a bias of only about 3.78 vehicles per video at a processing speed of 0.36 seconds. Based on these results, we highly recommend the user to use GPU if available to process videos and to use the model bolded below, which would be able to obtain statistics for roughly 660 cameras per day.

Detection model	Detection confidence threshold	Detection frequency (frames)	Tracking algorithm	Stop/start IOU threshold	Bias (vehicles)	Processing speed (sec)
yolov3-traffic	0.5	4	KCF	0.75	3.64	1.61
yolov3-traffic	**0.5**	**4**	**MOSSE**	**0.75**	**3.78**	**0.36**

VI. Deployment Approach

As stated in the introduction, we presented three deliverables to our partners:

- An open-source library that automatically collected live traffic video data, identifies vehicles and tracks their behavior, extracts descriptive statistics, and displays the results on a web application;
- A traffic dataset that estimated counts of vehicles and stop/start events at the junction-level, near real-time, in the ULEZ; and
- A web application that allowed users to conveniently access traffic data.

The open-source library was published on GitHub. We worked with our partners at Imperial College and the Turing Institute to make the library modular, so that they may customize it and increase its technical scope. The live analysis pipeline is hosted at Imperial College, and runs autonomously using existing AWS infrastructure (S3 bucket and EC2 instance). The traffic dataset is hosted on a PostgreSQL database supported by our partners at Imperial College. The web application is also hosted by our partners at Imperial College.

VII. Value Delivered

Per the project charter, our partners have defined value as an open-source library that automated the collection of live traffic video data and traffic induction data, and extracted descriptive statistics. With this library, project partners would be able to obtain real-time traffic statistics which are localized to the level of individual streets, easily and quickly. Our deliverables -- the open-source library, publically-available traffic statistics dataset, and web application -- exceeded this definition of value.

The project charter contained more specific goals regarding the final output of the delivered dataset. Concretely, our aim was to report vehicle counts, stop/start events, and speeds at near real-time for all cameras in the JamCam network. We were able to report counts and stop/start events, but unable to

compute speeds within the project timeline. Additionally, because we were limited by video processing speed, we were limited to reporting statistics for only vehicles within the ULEZ. However, our dataset still represented a major advantage over the current dataset, both in that it provided stop/start events and achieved fine-scale spatiotemporal resolution, with complete coverage in a region of London.

The deliverables unlocked opportunities for our partners along multiple axes to perform social good by reducing human exposure to air pollution from traffic. The dataset itself was to be used by researchers, including our partners, to understand traffic dynamics in central London and evaluate how traffic interventions might influence air quality and congestion. Our partners would easily visualize historical and real-time statistics on the web application, enabling them to identify, for example, how local traffic congestion could be linked road construction or improvements. The open-source library was made available to our partners so they could report data that specifically suits their needs, e.g., to include vehicles types not detected in our model, change algorithms that match their specific processing speed/accuracy. This data in turn would help our partners answer questions, such as the following that have already been proposed: what is the best bike path for bicyclists to travel to be exposed to the least air pollution? How might air pollution be attributed to traffic on a local or regional level? Additionally, the open-source library was convenient for researchers from other cities with similar camera infrastructure to make similar datasets to help them understand traffic dynamics, and how they might contribute to air pollution in other cities.

VIII. Limitations

The preeminent obstacle in this project was the tradeoff between coverage and accuracy in our delivered dataset. As described in the previous sections, model performance was measured along two axes, processing speed and accuracy. Given computing power constraints, we sacrificed coverage for accuracy. It is, however, possible with increased computing power to broaden the coverage to a wider network of cameras.

We were also unable to evaluate the performance of our model against real traffic flow data. This meant that while we were able to evaluate our model in its performance against annotated videos, we did not know if our model accurately predicts traffic dynamics. A comparison could be made between our data and lane-level induction loop data collected by TfL, or in-person recordings of traffic data at a junction. From this comparison, the following question could be evaluated: is 10 seconds of information per four minute interval on traffic counts on one arm of an intersection representative of traffic flow? This was a natural next step of the project.

Additionally, the existing pipelines were built on Amazon Web Services (AWS) infrastructure, and therefore, some reconfiguring would be required by the user to implement the library on a local machine rather than through AWS. Because of the computing limitations of a local machine, this would not be something we would recommend. However, should the user be interested in running the library locally, most of the pipeline should remain intact.

Finally, we have not tested whether our model has the potential to affect socioeconomically disadvantaged communities in London. One question that we had was whether the camera locations were geospatially correlated with identity markers, e.g., income level, racial makeup, age. A possible scenario could exist where because the JamCams are not evenly distributed along one of these markers, the reported traffic data may be biased against a disadvantaged group. Any interventions may therefore serve communities that have more traffic cameras, to the non-benefit or detriment of communities where traffic data is sparse.

IX. Next Steps

The next steps included evaluations of our current work and improvements to the technical scope of the project. Requisite evaluations were mentioned in the Limitations section, and include comparisons against traffic data and an analysis of potential biases induced by the coverage of our data. Additionally, a further step to demonstrate improvements in air quality model performance when implementing our dataset is an important point of evaluation to close the gap between our project motivation and our deliverables.

The breadth of future technical work was vast, and perhaps depended on how our partners needs evolve over time. Some potential avenues to be explored include:

- Evaluating and improving model performance at night, e.g., with transfer learning using night-time data
- Improving performance on London-specific data with transfer learning, e.g., for London buses, black taxis
- Implementing improved, custom-built tracking algorithms
- Estimating vehicle speed or acceleration in the videos
- Identifying pedestrians in the vehicle frame
- Measuring vehicle flow rather than counts per video
- Processing speed improvements

X. Appendix

Link to GitHub repo: https://github.com/dssg/air_pollution_estimation

X. a. Annotation Methodology

We used the following instructions to annotate ground-truth videos with the CVAT tool:

1. Use the video name as the task name so that it dumps with the same naming convention
2. As a rule of thumb, label things inside the bottom 2/3rds of the image. But if a vehicle is very obvious in the top 1/3rd, label it.
3. Use the following labels when creating a CVAT task:
    a. vehicle ~checkbox=parked:false ~checkbox=stopped:false @select=type:undefined,car,truck,bus,motorbike,van
4. Set Image Quality to 95

5. Draw boxes around key frames and let CVAT perform the interpolation
6. Press the button that looks like an eye to specify when the object is no longer in the image
7. Save work before dumping task.