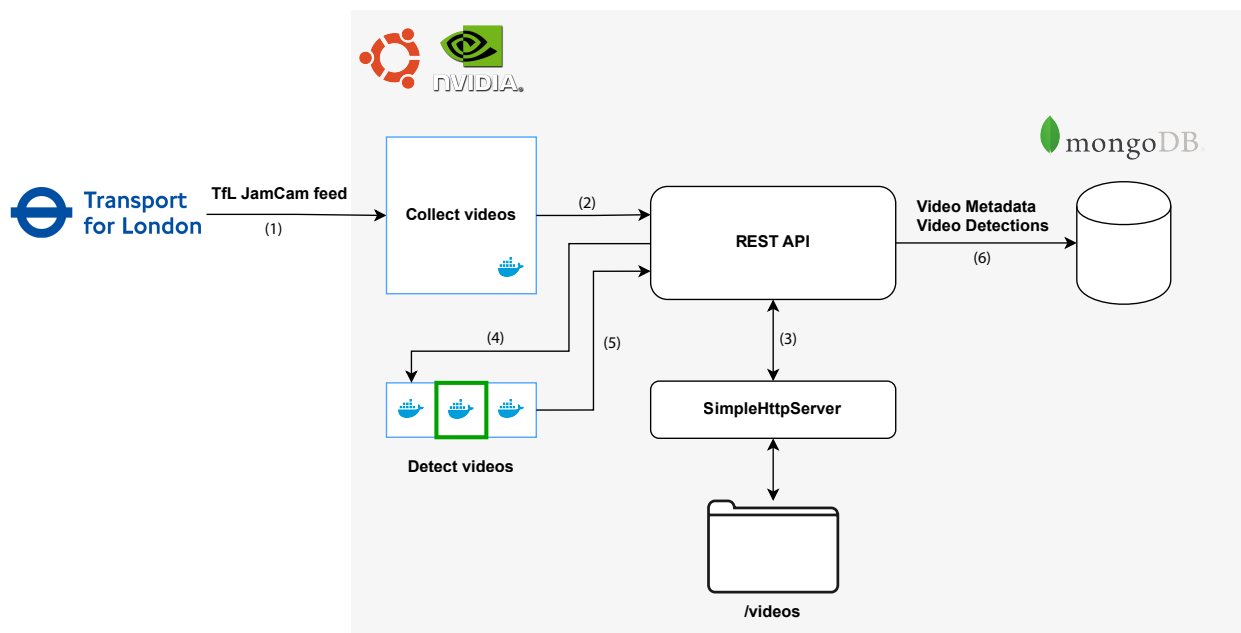


# Graphical Abstract

## Scalable object detection pipeline for traffic cameras: Application to TfL JamCams

Huan Min Gan, Senaka Fernando, Miguel Molina-Solana



## Highlights

### **Scalable object detection pipeline for traffic cameras: Application to TfL JamCams**

Huan Min Gan, Senaka Fernando, Miguel Molina-Solana

- Scalable real-time video detection pipeline using traffic cams.
- Built upon YOLOv3 and Docker.
- Results in London are consistent with existing traffic patterns.

# Scalable object detection pipeline for traffic cameras: Application to TfL JamCams

Huan Min Gan<sup>a</sup>, Senaka Fernando<sup>a</sup> and Miguel Molina-Solana<sup>a,b,\*</sup>

<sup>a</sup>Dept. Computing, Imperial College London, UK

<sup>b</sup>Dept Computer Science and AI, Universidad de Granada, Spain

## ARTICLE INFO

**Keywords:**  
traffic detection  
video processing  
TfL JamCam  
YOLO

## ABSTRACT

With CCTV systems being installed in the transport infrastructures of many cities, there is an abundance of data to be extracted from the footage. This paper explores the application of the YOLOv3 object detection algorithm trained on the COCO dataset to the Transport for London's (TfL) JamCam feed. The result, open-sourced and publicly available, is a series of easy to deploy Docker pipelines to create, store and serve (through a REST API) data on identified objects on that feed. The pipelines can be deployed to any Linux machine with an NVIDIA GPU to support accelerated computation. We studied how different confidence thresholds affect detections of relevant objects (cars, trucks and pedestrians) in London JamCam scenes. By running the system continuously for 3 weeks, we built a dataset of more than 2200 detection datapoints for each camera (~6 datapoints an hour). We further visualised the detections on an animated geospatial map, showcasing their effectiveness in identifying traffic patterns typical of an urban city like London, portraying the variation on different object population levels throughout the day.

## 1. Introduction

Much of society's economy requires its transport networks to move both people and goods (Wootton et al., 1995). Traffic congestion instances occurs frequently on these transport networks, affecting the overall utility of a road while unnecessarily increasing fuel consumption and air pollution (Chin, 1996; Wang et al., 2012). Proposals to address traffic congestion need to be conceived and adopted quickly—and in many occasions in near real-time—due to hugely varying mobility demand, yet they are limited by expensive and quickly outdated traffic survey data (Strategic Transportation & Department of the Built Environment, 2018).

For instance, as of 2018, the City of London borough have held the Traffic Composition Survey (TCS) every 2 years to quantitatively measure London's traffic makeup. This survey is conducted by positioning humans at select locations for a period of 2 weeks, to count private cars, taxis, motorcycles, light goods vehicles, heavy goods vehicles, public service vehicles, bicycles, and pedestrians. The data from this survey is then used to extrapolate trends across sample years and the entire city. While this method does provide some insight into the London's traffic composition trends, it is severely limited. Firstly, the data produced is restricted to the selected two week period where the survey was conducted and is therefore unable to measure changes in traffic composition throughout the year. Secondly, the count locations are confined to the City of London borough and thus no definitive data about the traffic composition across the entire London exists. On the whole, the TCS is manually intensive, provides a limited view of the traffic situation, and is expensive to carry out, requiring human labour for extended periods.

This example highlights that existing data on traffic congestion does not accurately reflect the present-day situation on traffic networks but just a historical view of the past. Hence, there is a need to produce real-time data for traffic congestion which in turn can inform and support dynamic management of transport networks.

Transportation departments around the world have indeed deployed traffic cameras across cities and major roadways to regularly observe vehicular traffic throughout all times of the day. Many also link the video feed to the internet, allowing commuters to view current traffic conditions (e.g. (allroadcams, 2020; Transport for London, 2020a; Transport for NSW, 2020)). Websites showing the traffic camera feed either stream small snippets of video or still images. This information is widely considered to be a part of the public domain. In London, for instance, Transport for London (TfL)

\*Corresponding author

✉ huan.gan18@alumni.imperial.ac.uk (H.M. Gan); senaka.fernando15@imperial.ac.uk (S. Fernando); mmolinas@ic.ac.uk (M. Molina-Solana)

ORCID(s): 0000-0003-1290-9118 (S. Fernando); 0000-0001-5688-2039 (M. Molina-Solana)

offers its traffic camera (JamCam) feed under an open data policy aimed to make these public data freely accessible to third parties (Transport for London, 2020c). The cameras providing the video feeds are strategically placed along major roads in Greater London and highly trafficked areas in Central London, making it a reliable data source for assessing the traffic situation in London.

Recent computational advancements in object detection systems, such as YOLOv3 (Redmon & Farhadi, 2018), now make it possible to identify different objects in a scene at close to real-time rates (Zhao et al., 2019). Some studies (as discussed in Section 2.1) have applied these developments to carry out object detection on traffic scenes, proving it is possible to detect traffic objects to a high level of accuracy on low resolution images typically produced by network traffic cameras. However, these studies do not produce efficient, self contained pipelines which can be easily deployed for practical purposes. In this paper, we propose a scalable and distributed data processing pipeline which consumes London's TfL JamCam feed and applies an off-the-shelf YOLOv3 object detection algorithm, trained over the COCO dataset (Lin et al., 2014), to continually generate a dataset of traffic objects detected from the feed. The approach of using YOLOv3 pre-trained over COCO can also be found in the works by Oltean et al. (2019) and Lin & Sun (2018). We put forward a production-ready solution—tailored for three *objects of interest (OoI)*: cars, trucks and pedestrians—which can be deployed onto a single Linux machine with an NVIDIA GPU installed. We tested and validated this setup to efficiently process the video feed provided by TfL, persistently storing data about videos and the objects detected.

The rest of this manuscript is organized as follows: Section 2 discusses existing works on object detection on videos, with a particular interest on traffic videos. Section 3 describes the design and implementation of our proposed solution and Section 4 demonstrates its particular application to London. The work ends with some conclusions and pointing towards further actions.

## 2. Background

Object detection is a computational technique to estimate the location and class of different objects within an image (Zhao et al., 2019). These algorithms are currently used in a wide range of scenarios such as self-driving cars, license plate reading, and facial recognition. When several objects need to be detected in an image, the term multi-object detection is often used (Redmon et al., 2016; Tao et al., 2017). The approach to multi-object detection in computer vision performs two distinct tasks: determining what objects exists in an image (classification) and the location of each object (localisation, a regression task).

Current state-of-the-art models for object detection include, among others:

**R-CNN (Girshick et al., 2014)** It combines region proposals with convolutional neural networks (CNN), hence the name. 2000 regions are proposed through a selective search algorithm (a greedy algorithm which aims to combine similar regions into a larger region). Each region proposal is then fed as input to a CNN to extract a feature vector, which is subsequently inputted to a support vector machine (SVM) to classify the object in the region proposal.

**Fast R-CNN (Girshick, 2015)** An iteration of R-CNN in which the 2000 proposed regions are not fed to the CNN but the input image itself is, to extract a feature map. From the feature map, the region of proposals are identified using selective search and then fed to a classifier. This method is faster than the original R-CNN, and getting region proposals using selective search is the test-time computational bottleneck.

**Faster R-CNN (Ren et al., 2017)** The third iteration by Girshick replaces selective search in Fast R-CNN with a region proposal network (RPN). RPN is applied on the output feature map from the CNN to generate region proposals which are then fed to a classifier. This iteration is reported to improve test-time speed by a magnitude of 10.

**Mask R-CNN (He et al., 2017)** This model, an extension of Faster R-CNN, provides semantic segregations (i.e. outlines) of objects in images. Mask-RCNN implements a mask branch on top of the Faster-RCNN output to segregate the object from the image.

**R-FCN (Dai et al., 2016)** The region based fully convolutional neural network (R-FCN) adopts a similar two stage detection strategy of region proposal and object classification of the region. Candidate regions are extracted by an RPN and fed through the R-FCN architecture which classifies the regions into object classes or background.

**Table 1**

Studies which have applied different object detection models on traffic scenes

| Paper                                                                                                                   | Published | Models used                                     | Image Resolution      |
|-------------------------------------------------------------------------------------------------------------------------|-----------|-------------------------------------------------|-----------------------|
| Effective Object Detection From Traffic Camera Videos (Shi et al., 2017)                                                | Aug 2017  | Faster-RCNN, R-FCN                              | 1080x480<br>1920x1080 |
| An Object Detection System Based on YOLO in Traffic Scene (Tao et al., 2017)                                            | Nov 2017  | Fast-RCNN, Faster-RCNN, R-FCN, SSD, YOLO, OYOLO | 448x448               |
| A Deep Learning Approach of Vehicle Multi-target Detection from Traffic Video (Li et al., 2018)                         | Nov 2018  | YOLO-voc, YOLO 9000, YOLOv3, YOLO-vocRV         | 300x225<br>550x448    |
| Traffic Congestion Detection from Camera Images using Deep Convolution Neural Networks (Chakraborty et al., 2018)       | Dec 2018  | YOLO, DCNN, SVM                                 | 400x225<br>416x416    |
| Large-Scale Object Detection of Images from Network Cameras in Variable Ambient Lighting Conditions (Tung et al., 2019) | Dec 2018  | YOLO                                            | 1080x480<br>1920x1080 |

R-FCN's fully convolutional architecture is considered more efficient as computation is shared across the whole image.

**SSD (Liu et al., 2016)** Known as single shot detectors, this model takes a different approach to object detection. It first inputs the image to a CNN to compute a feature map. Bounding boxes are defaulted to different aspect ratios and scales according to the feature map. At test time, SSD predicts scores for each object category it was trained to predict in each default box and adjusts the bounding box to align with the detected object.

**YOLO (Redmon et al., 2016)** It frames object detection as a regression problem, applying one neural network architecture at test time. It first divides an image to a grid of cells, where each cell predicts a fixed number of bounding boxes and class probabilities for the objects within the boxes. Thresholding the results above a certain value gives the final object detections.

For a more in-depth review of the different algorithms and a comparison between them, we refer the interested reader to the works by Zhao et al. (2019) and Liu et al. (2020).

## 2.1. Detecting objects on traffic scenes

Object detectors have been applied to traffic scenes across various cities around the world. Table 1 lists some of those studies, together with a brief overview of the used detector models and the range of images they operate on. In these papers, the base YOLO model (Redmon & Farhadi, 2018) is consistently mentioned as a reference point to any improvements to traffic object detection. The main reason for that is YOLO being fast enough for real-time object detection, a massive advantage when coupled with a live video feed as it opens up potential for real-time systems. These studies consistently demonstrate that object detection of traffic objects using YOLO is feasible at resolutions as low as 300 x 225 pixels.

The base YOLO model was presented as being extremely fast and able to process images at high speeds of 45 frames per second (Redmon et al., 2016). However, it was less accurate than other object-detection systems, struggling to precisely localise objects. YOLOv2 and YOLOv3 solve these issues and are major improvements to the original YOLO model. YOLOv3 is nowadays considered to be a state-of-the-art detector (as of February 2020) and its performance in comparison to other popular object detection methods has previously been benchmarked. Figure 1 shows that YOLOv3 achieves higher mAP (mean Average Precision) scores on the COCO dataset (Lin et al., 2014) with a faster inference time as compared to other object detectors (Redmon & Farhadi, 2018).

Works in Table 1 presented several approaches for traffic detection. Shi et al. (2017) and Tao et al. (2017) found that the RCNN architecture performs better at fine-grained object classification of an object, though with a limited speed. The aim of Chakraborty et al. (2018) was to classify the level of congestion of an image, by classifying the entire image as being congested or non-congested, and training object detection architectures on these images. YOLO is found to compute faster than all other models and has the highest accuracy, precision and recall. Li et al. (2018)

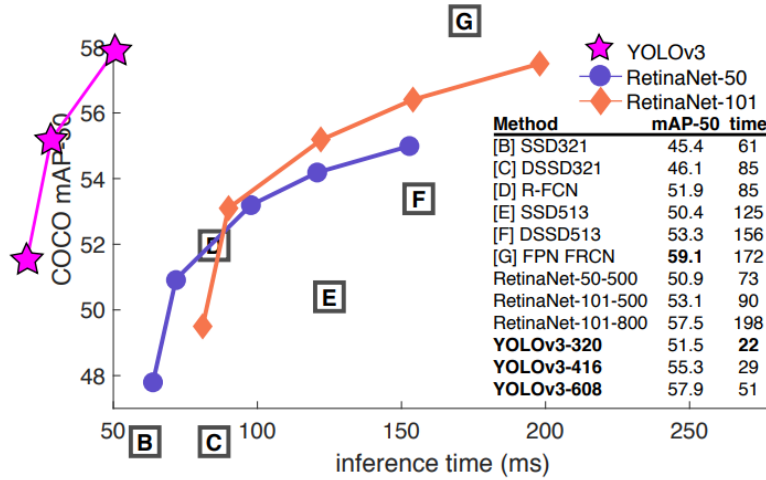


**Figure 1:** Performance comparison of object detection models, evaluated against its mAP and inference time (respectively representing how accurate and fast the model detects on a single image). Measures were obtained using a GeForce GTX TITAN X GPU. Image courtesy of Redmon (Redmon & Farhadi, 2018).

**Table 2**

System requirements for object detection in traffic cameras

| Requirement Type | Description                                                                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Functional       | The detection system must output reasonable count estimates for objects in view.                                                                                             |
| Functional       | The REST endpoint providing the video feed may not provide a correct datetime stamp in its JSON response. The built solution should therefore provide a reasonable estimate. |
| Functional       | In some cases, there is no way of knowing if the provided video has been previously seen. The system must therefore differentiate between repeated feeds.                    |
| Functional       | The cameras often go offline for maintenance, the system should detect this.                                                                                                 |
| Non-functional   | The built system must be practical to deploy for further, continued use, so it can be left running to gather video and detection data for extended lengths of time.          |
| Non-functional   | The built system should be built to be easily scaled for speed or volume.                                                                                                    |

trained a model based on YOLO to achieve high accuracy in the three traffic states (free flow, synchronous flow and blocking flow). It addresses YOLO's known weakness at identifying all the objects in an image when many objects are present. Tung et al. (2019) precisely highlights those limitations of the YOLO model by showing how it struggles to consistently detect the same humans and cars as their positions move from one frame to the next, and how it outputs a higher rate of false positives for detecting cars at night (because streetlamps are often misidentified as cars).

As a conclusion, we can say that several functional and non-functional requirements are necessary to be implemented in order to have a successful object detection system in traffic scenes. They are listed in Table 2.

In addition to those requirements, there are some specific challenges when working on this domain. Among others, the main ones are 1) the only availability of low resolution videos, 2) images being blurry due to dirt or vibration, and 3) relevant objects being far away, oddly shaped (due to perspective) or partially occluded.

### 3. System design and implementation

Those requirements for a real-time traffic detection system led us to propose the architecture illustrated in Figure 2. It comprises a module for collecting videos, a module to detect objects in those videos, a REST API to interact with the data, a MongoDB database, and a file storage system. We decided to focus on three of the most common objects in traffic scenes: cars, trucks and pedestrians. They will constitute the Objects of Interest (OoIs) of our system.

We chose MongoDB as the database because it enables fast read and write operations (Kabakus & Kara, 2017), and it does not require a pre-fixed table structure, which fits nicely with object detection problems, in which the classes

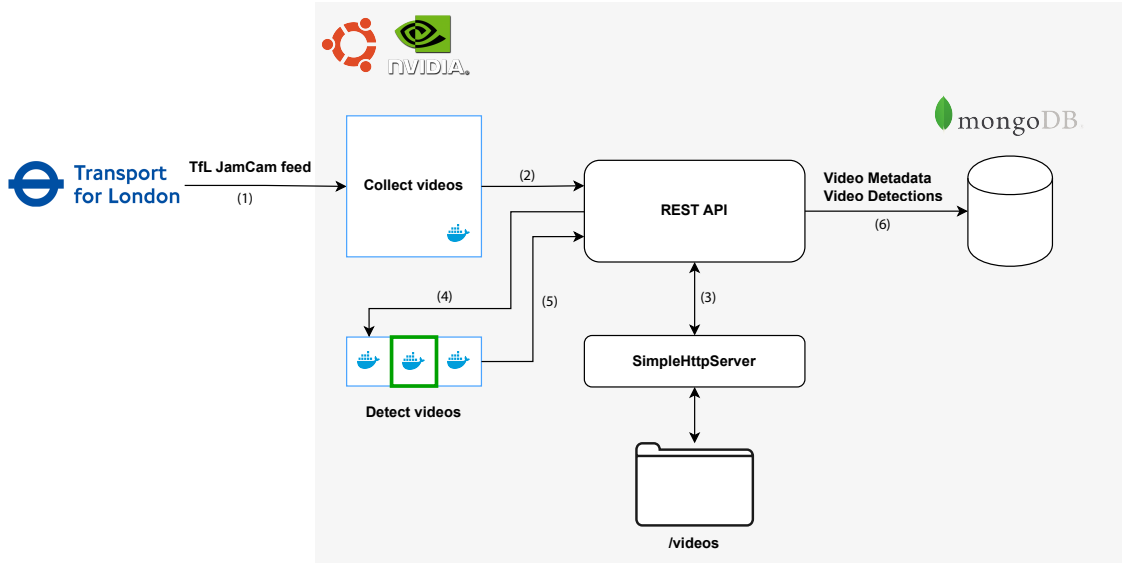


**Figure 2:** System architecture of the deployed system onto a single machine. The Docker container pipelines, Flask REST API, MongoDB and a SimpleHttpServer emulating the host filesystem are hosted on a Linux machine with NVIDIA GPU. Video files are stored in the /videos directory on the host system. The workflow is explained in Section 3.3.

and quantity of objects in an image might not be known in advance. Flask is used to build the REST API that controls access to file storage and the database and that exposes the video and detection information.

The data processing pipelines, built with containerised python scripts, are designed to process the most recent videos from all available cameras per container lifecycle. This is to ensure the Docker container does not run for extended periods of time. Docker containers provide a solution to reliably run software in different computing environments. Each container installs just enough of its dependencies within its container environment to execute its service and can be run on the Docker engine application, which in turn can be installed on many different operating systems (Merkel, 2014).

We chose to containerise the service consuming camera feeds, making it possible to accommodate more cameras by simply deploying more containers. This modular approach allows scaling the system at the individual container level rather than the entire pipeline (we aimed at optimizing GPU usage). The implementation of this system (with instructions on how to use it) is available at <https://github.com/ministrudels/JamCam-Detector>.

### 3.1. Data Collection and preprocessing

This module exists to get the latest video feeds from the provider and store them in a permanent video storage location. It consists of a single container which processes a range of feeds (specified in its arguments). Several instances of this container can be run in parallel in order to quickly gather all new videos. First, an estimate of the video file's timestamp and checksum are generated, fulfilling the 2nd and 3rd design requirements (associating a timestamp to the video file and detecting a fresh video feed). If the checksum has not been yet seen by the REST API (meaning this is a new video), the video file and its metadata is subsequently sent for storage.

It would be at this stage where any further preprocessing of the collected videos might happen, such as resizing them or adjusting their contrast. Other issues that might need to be tackled here are how to deal with blocked or dead cameras (i.e. cameras that do not offer a proper video), and cameras that have overlapping views (so objects are not counted several times).

In our particular situation (as we will describe below), and with the aim of demonstrating how easy is to deploy an automatic detection system for traffic cameras, we do not perform any resizing or color adjustment, inputting the JamCam videos directly (as they come) to the detection pipeline. Similarly, we do not identify in advance scenes in which the view is blocked; we feed them to YOLO expecting that it returns zero objects (indeed, that is what happen as we will demonstrate in Section 4.1).

Finally, due to the nature of JamCams in London, overlapping views are extremely rare: traffic cameras are strate-

gically placed with sufficient distances away from each other to cover as many junctions as possible. Thus, we assumed this situation does not occur in London, and made no provision to account for it.

All in all, while it is certainly important to be as accurate as possible in detecting objects, the main objective of our system (as it is when a human operator aims to detect traffic) is to offer approximate counts at different locations and times. In that regard, it is not highly relevant, for instance, whether 15 or 16 cars were identified in a particular scene.

### 3.2. REST API

As mentioned earlier, we created a simple REST API (in Flask) to create, read and update registers in both the database storage and the file storage. It exposes information about video metadata and object detections. We designed the API to mimic that from the provider. It also provides a checksum value for the latest video stored in the database and file storage system, as well as the URL it is available at. The response object contains an element per camera. To optimize its response time, the API regularly proactively caches a response object with the latest video checksums and endpoint urls. This allows the API to quickly serve the cached response to incoming GET requests on a given object.

### 3.3. Detecting objects in videos

The module to detect objects in videos (Figure 3) performs object detection on the stored videos and posts the detection information back to the REST API to store these data in Mongo. It is a three staged process packaged in Docker containers with a shared volume to share video files between containers.

The *input frame container* (1) extracts one frame per second from the stored videos using FFMPEG and sends a filepath string through a socket connection to the *gpu\_darknet*'s (2) STDIN stream. It also determines if a video is offline through its video length (videos shorter than 2 seconds are considered errors). It targets a specified range of feeds in its arguments to facilitate scale. The *gpu\_darknet* container reads filepaths to images through its STDIN and outputs the results through STDOUT giving its object predictions, each with a confidence score and bounding box information. Container (3) parses darknet logs by reading the STDOUT stream of the *gpu\_darknet* container to calculate the number of detections. It does this by aggregating the detections of each input frame from a camera and averaging them. The resulting JSON detection object is then passed back to the REST API to be subsequently stored in mongoDB.

The `./darknet` command is packaged to accept filepaths to images or videos from STDIN and outputs its detections and inference time to STDOUT. Here, a design decision was made to encapsulate the detector in its own Docker container to quickly be able to apply it and only worry about interacting with its STDIN and STDOUT through python scripts. In order to create a portable detector, nvidia Docker is used to "enable deployment of GPU-accelerated applications across any Linux GPU server" (Olson et al., 2016). With this approach, the detector is being used "out-of-the-box" in a portable container and may be configured through its ENTRYPOINT command in the Dockerfile.

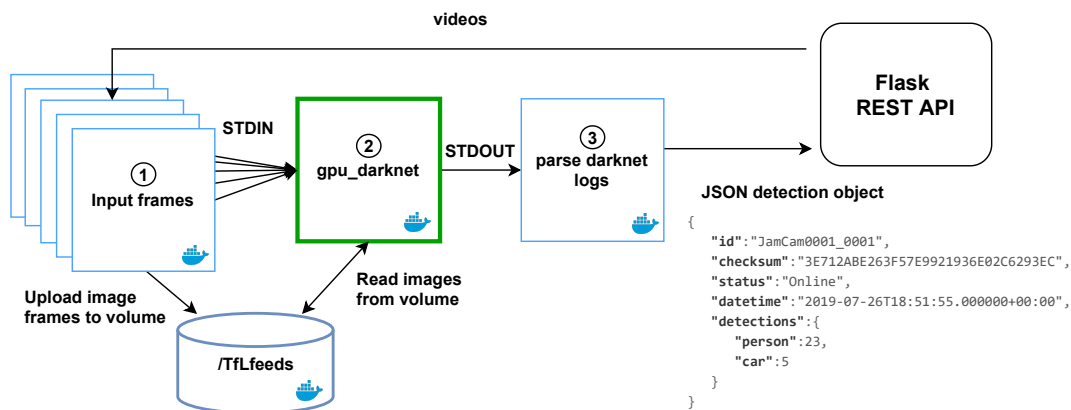


**Figure 3:** Detection pipeline built with Docker containers: (1) an *input frames* container, (2) a GPU-optimised YOLOv3 container (using the darknet image), and (3) a *parse logs* container. The final output is a JSON file containing information about the detected objects.

We tested the object detection pipeline in four Nvidia GPU models (Quadro M4000, Tesla V100, Tesla P100, and GeForce GTX 1070). On these, we found that the original YOLOv3 repository ([github.com/pjreddie](https://github.com/pjreddie)) did

not support all GPU architectures: the output was either empty or very slow for Nvidia Quadro M4000 and Tesla V100. Because of that, we selected AlexeyAB's implementation of YOLOv3 (an extension to Redmon's original implementation), written in C and CUDA and supporting accelerated GPU computation<sup>1</sup>. It adds further optimisations to the original YOLOv3 darknet model and contains additional features (as reported in the release's README) which makes it a practical option. Among the key features relevant to this project are: 1) a faster neural network architecture which exhibits 3 times faster detection times and 2 times faster training times on GPUs with Tensor Core architecture as compared to the original YOLOv3 repository; and 2) a useful flag option to get more types of output from the darknet model for debugging purposes, like saving the detection video file.

We published these different `gpu_darknet` images to DockerHub to allow quick building under the tags `ganhuanmin/gpudarknet` and `ganhuanmin/gpudarknet:tensor_core`.

## 4. Experimental set-up: the London case

In order to validate our architecture proposal, we applied it to London's TfL JamCam footage. The TfL open data policy aims to make freely accessible to third parties the public data TfL collects (Transport for London, 2015). TfL's open data is also used by companies such as Waze, Google, Apple and Citymapper (Deloitte, 2017) so its quality is well established. Traffic camera feeds from TfL's JamCams are provided under this open data policy and is served over the Unified API, a REST-like API responding with JSON objects (Transport for London, 2020c). Currently, TfL staff observe the footage and assess traffic congestion levels. This sometimes involves remotely operating specific cameras to get a better view.

At the time of writing (summer 2020), there are over 700 different TfL traffic cameras which update the feed every three minutes (or less) (Transport for London, 2020b). The feed provides fresh nine-second videos at 25 FPS and CIF resolution (352 x 288 pixels), sufficient quality to perform object detection on, as explained in Section 2.1. These videos are available to download via a public AWS S3 bucket and are overlaid with the TfL logo, locale, and date time stamps. Some camera's images are blurry due to dirt and vibration from incoming traffic. In summary, the dataset from TfL provides a robust experimental setup, providing low resolution video feed, real-world image quality issues, and cases where feeds are offline yet unreported.

To preliminary assess the quality of our detection pipeline (using off-the-shelf YOLOv3 trained on COCO dataset), we run it on a set of JamCam videos, overlaying the numbers of detections on the TfL video feeds (an example of which is shown in Figure 4). As mentioned in Section 1, we limit our *OoIs* (Objects of Interest) to cars, trucks and pedestrians, discarding any other objects (such as backpacks, parking meters, stop signs, fire hydrants, etc.) that this YOLO model is able to detect. For a human, it is difficult to reasonably identify all objects in a scene with great confidence due to low image resolution. In these scenes, *OoIs* identified by a human are more informed by contextual information of the scene (i.e. if the individual objects were cropped from the image and displayed alone, it would be extremely difficult to identify them). Conveniently, YOLO is able to use the entirety of an image to inform the classification of an object (Redmon & Farhadi, 2018).

The implementation of the YOLOv3 model allows for adjusting the detector's sensitivity by means of lowering or increasing the confidence score for detections (Redmon & Farhadi, 2018). This means that only those detections with a confidence score over the threshold will be reported as such, and offers the possibility of having a more fine-tuned detection system by adjusting this confidence score to different image sets and *OoIs*.

Table 3 summarizes how well the COCO-trained YOLOv3 model is able to detect cars, trucks and pedestrian on a set of videos collected from the TfL video feed. Counts are given for three different confidence thresholds (25%, 10% and 5%). A manual count of relevant objects (*Ground Truth* column) is also reported.

### 4.1. Identifying different scenes to improve detections

A closer inspection of the results behind Table 3 indicated that around 1/20 of the cameras showed blatantly wrong counts. An additional issue regarding London's JamCam feeds is the different vantage points of the cameras. Our early assessment on the results indicated a trend where the detector performed poorer on cameras placed at higher vantage points.

To account for this behavior, we decided to manually segment the video feeds by the perceived distance to the *OoIs* and subsequently aimed at finding the most adequate confidence threshold for each set. The different sets of images

<sup>1</sup>Available at [https://github.com/AlexeyAB/darknet/releases/tag/darknet\\_yolo\\_v3](https://github.com/AlexeyAB/darknet/releases/tag/darknet_yolo_v3)

**Table 3**

Detections of relevant objects by YOLOv3 for different confidence threshold levels

| Object      | Ground Truth | Confidence Thresholds |     |     |
|-------------|--------------|-----------------------|-----|-----|
|             |              | 25%                   | 10% | 5%  |
| Cars        | 187          | 164                   | 219 | 265 |
| Trucks      | 9            | 17                    | 41  | 52  |
| Pedestrians | 259          | 70                    | 180 | 228 |

**Figure 4:** Two JamCam images with the (cars, trucks & pedestrian) counter overlay in their bottom-right corners.

are described in Table 4 and represent specific types of TfL JamCam scenes whilst containing a sufficient number of *OoI*s.

In this context, we considered an *OoI* to be ‘far’ away only if the cropped image of the object is difficult to classify by the human eye. On the other hand, an *OoI* is considered ‘close’ if the cropped image of the object is easy to classify. We also compiled a specific set, at another point in time, for trucks (because the London Lorry Control Scheme (LLCS) prohibits movement of heavy goods vehicles during certain times in the week (London Councils, 2020)).

**Table 4**

Types of JamCam scenes

|       |                                                       |
|-------|-------------------------------------------------------|
| Set A | Scenes where <i>OoI</i> are ‘far’ away                |
| Set B | Contains both ‘far’ <i>OoI</i> and ‘close’ <i>OoI</i> |
| Set C | Scenes where <i>OoI</i> are ‘close’                   |
| Set D | Scenes with no <i>OoI</i> or blocked                  |
| Set E | Scenes with trucks                                    |

Images from a single TfL camera may be classified under sets A, B, C and D throughout the day as JamCam operators often move and zoom the cameras. From observing a sample of image sets, the proportion of TfL feed images belonging to sets A, B, C and D were approximately 2%, 70%, 23%, 5% respectively. Since JamCams are affixed to its vantage points, the views do not change significantly and thus we assumed these proportions of images reasonably represent JamCam images at all times.

The results of applying different confidence score threshold are summarised in Tables 5, 6 and 7. A weighed version of the Mean percentage error (MPE), shown in Equation 1, is calculated for each object’s detection accuracy across sets A, B and C for the different threshold values across each set. Set D (scenes with no *OoI* or blocked) is purposely excluded from the MPE calculation as there are no ground truth objects here, though we include it in the tables to observe whether unexpected detections are obtained from these videos. The MPE metric is used to get an idea of overall detection error on all JamCam feeds, giving fair weighting to the most common situations. Here,  $f_i$  is the number of objects detected by the model,  $a_i$  is the ground truth (real number of objects in the scenes), and  $w_i$  is

the weight (proportion of that set from the total).

$$MPE = \frac{100\%}{\sum w_i} \times \sum_{i=1}^n w_i \frac{f_i - a_i}{a_i} \quad (1)$$

Looking at the three tables, we can observe that the number of detections for Set D is mostly zero. This is the expected behavior because Set D contains videos in which the image is blocked or not pointing towards a road. Thus, we can be confident that the system will not provide unreliable results on videos not containing traffic scenes.

MPE for car and truck objects both increase when a lower confidence threshold is applied (a lower threshold ultimately means that more detections will be reported, including false positives). This is primarily because of the proportion of images belonging to sets B and C, which make up an estimated 93% of images the JamCam sees. The images representing sets B and C also contain crowded scenes, a situation YOLO does not perform well in. Having said that, in all sets A, B, C and E the default threshold value of 25% on cars exhibits the lowest MPE in comparison with other thresholding values.

Regarding trucks, the lack of enough truck objects across our original sets A, B and C (due to the London Lorry Control Scheme (LLCS) being active in the set of videos we collected) posed a challenge. To overcome this, we specifically created set E with examples at another time point. These examples in set E could also be classified as belonging to B and C (but just at another time point). Therefore, Table 6 shows that a thresholding value of 25% for trucks is sufficient for 93% of JamCams.

Table 7 confirms that applying a lower thresholding score for detecting pedestrian objects in London yields a much higher accuracy. This is particularly true for sets A and B, where pedestrians are far away and it is thus difficult to reasonably identify them with high confidence.

**Table 5**

Detections of car objects with different confidence threshold levels. Set D is excluded from the MPE calculation and only included for checking purposes.

|         |     | Ground Truth    | Confidence Thresholds |      |      |
|---------|-----|-----------------|-----------------------|------|------|
|         |     |                 | 25%                   | 10%  | 5%   |
| Set A   | 2%  | 86 <sup>2</sup> | 69                    | 103  | 124  |
| Set B   | 70% | 59              | 54                    | 66   | 85   |
| Set C   | 23% | 42              | 41                    | 50   | 55   |
| Set D   | 5%  | 0               | 0                     | 0    | 1    |
| Set E   | N/A | 19              | 25                    | 41   | 50   |
| MPE (%) |     |                 | -7.2                  | 13.8 | 40.9 |

**Table 6**

Detections of truck objects with different confidence threshold levels. Set D is excluded from the MPE calculation and only included for checking purposes.

|         |     | Ground Truth | Confidence Thresholds |       |       |
|---------|-----|--------------|-----------------------|-------|-------|
|         |     |              | 25%                   | 10%   | 5%    |
| Set A   | 2%  | 3            | 5                     | 9     | 14    |
| Set B   | 70% | 3            | 5                     | 15    | 18    |
| Set C   | 23% | 3            | 7                     | 17    | 20    |
| Set D   | 5%  | 0            | 0                     | 0     | 0     |
| Set E   | N/A | 25           | 20                    | 29    | 36    |
| MPE (%) |     |              | 82.8                  | 411.9 | 513.3 |

<sup>2</sup>Ground truth is an approximate here as it is extremely difficult to identify the correct counts of objects in these images.

**Table 7**

Detection of pedestrian objects with different confidence threshold levels. Set D is excluded from the MPE calculation and only included for checking purposes.

|         | Set Proportion | Ground Truth     | Confidence Thresholds |       |     |
|---------|----------------|------------------|-----------------------|-------|-----|
|         |                |                  | 25%                   | 10%   | 5%  |
| Set A   | 2%             | 80 <sup>2</sup>  | 16                    | 43    | 66  |
| Set B   | 70%            | 170 <sup>2</sup> | 45                    | 124   | 144 |
| Set C   | 23%            | 9                | 9                     | 13    | 16  |
| Set D   | 5%             | 0                | 0                     | 0     | 2   |
| Set E   | N/A            | 17               | 18                    | 45    | 58  |
| MPE (%) |                |                  | -55.9                 | -10.2 | 7.2 |

## 4.2. Adjusting the detector's confidence score for different objects

Across all sets A, B, C, D and E, lowering the threshold score increases the rate of false positive detections. This effect is even more pronounced in crowded images, where many *OoIs* are within the scene. However, it was also observed that lowering the threshold score to 5% significantly increases the detector's performance in scenes where *OoIs* are 'far' away (as seen in sets A and B). In particular, lowering the threshold significantly increases YOLO's ability to detect pedestrians while minorly increasing its ability to detect cars where these objects are 'far' away. False positives occur more frequently in scenes which are densely populated.

This experimental study demonstrates that lowering the detection threshold on YOLO allows it to detect objects which are further away at the cost of increasing false positive detections on the whole. However, we should take into account that in a system devoted to identify traffic, it is often preferable to report a bit more traffic than real (false positives are less costly) rather than less.

On the light of all these observations, we decided on selecting different threshold scores for different objects. As said, lowering the threshold score from the default 25% increases false predictions of trucks as car objects. A score of 25% was therefore chosen for trucks and vans to reduce its false positive detections, as it seemed trucks have a higher propensity of being falsely identified. For cars, on the other hand, we decided on using 10% at the expense of more false positives. This is because in this scenario, it is better to minimise the number of false negative detections over minimising false positive. We observed that false positives for cars occurs frequently in crowded scenes, and hence increasing the confidence threshold provides better accuracy in these situations. Finally, we chose 5% for the pedestrian detector in order to maximise the ability to recognise people further away.

In summary, trucks: 25%, cars: 10%, and pedestrians: 5%.

## 5. Results and discussion

With the necessary pipelines built, validated and adjusted, we run the system for three weeks gathering a dataset comprising 2,258,145 mp4 videos (~300 GB worth of videos), subsequently detecting the *OoIs* in them. For visualisation purposes, we plotted the detections on a geospatial map using geopandas, matplotlib and cartopy. Animating this on a map against time gives an idea about how a certain object's presence in London streets changes over time. Two types of plots were made for each *OoI* following the below formulas, where  $I$  is the colour intensity,  $t$  is the datapoint in time, and  $x$  is the count of *OoI*:

**Raw:** Intensity  $I_t$  of a map point represents the count  $x_t$  of *OoIs* at the camera at that position.  $I_t = x_t$

**Normalised:** Intensity  $I_t$  of a map point represents the relative 'business' of the camera at that position.  $I_t = \frac{x_t}{x_{max}}$

### 5.1. Detecting Cars

Cars are the most prevalent object seen in TfL's JamCam feeds. Figures 5 and 6 show the relative level of detected car objects for both within the M25 (the major motorway encircling Greater London) and inner city London. Since cars are the most common object seen on the road, it can be argued that its count serves as an indicator of traffic levels. Normalising the count (using each camera's maximum seen count) lets us ignore whether or not the camera is placed

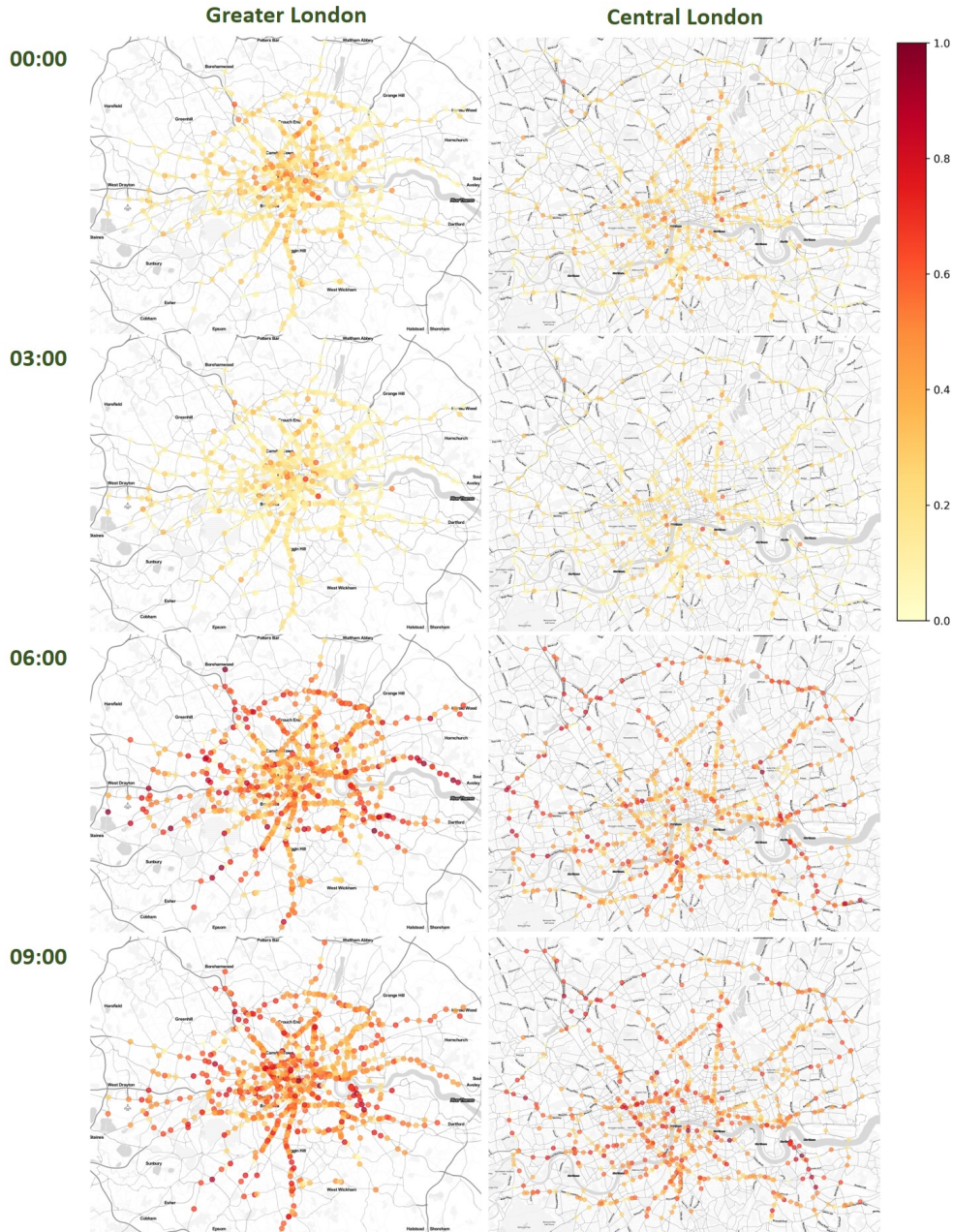


**Figure 5:** Normalised level of cars within the M25 (left) and inner city London (right) throughout the first half of Wednesday 31 July 2019. The M25 is the major motorway encircling Greater London

on a higher vantage point (cameras on a higher vantage point have a larger field of view and thus will see more OoI; by normalizing the count, we can then see a camera's relative busyness). The datapoint can then be taken to represent what proportion of cars the camera observes relative to the maximum number of cars it has ever seen for the period (3 weeks in our experimentation).

Traffic congestion typically peaks twice in the day, early before working hours and after working hours when the population is commuting to and from work. The figures illustrate how traffic follows a noticeable pattern of quickly becoming busy early in the day and taking a longer period to dissipate to empty levels. This observation is consistent with the study conducted by TfL, which found the peak spreading phenomenon to be apparent on London roads. Peak

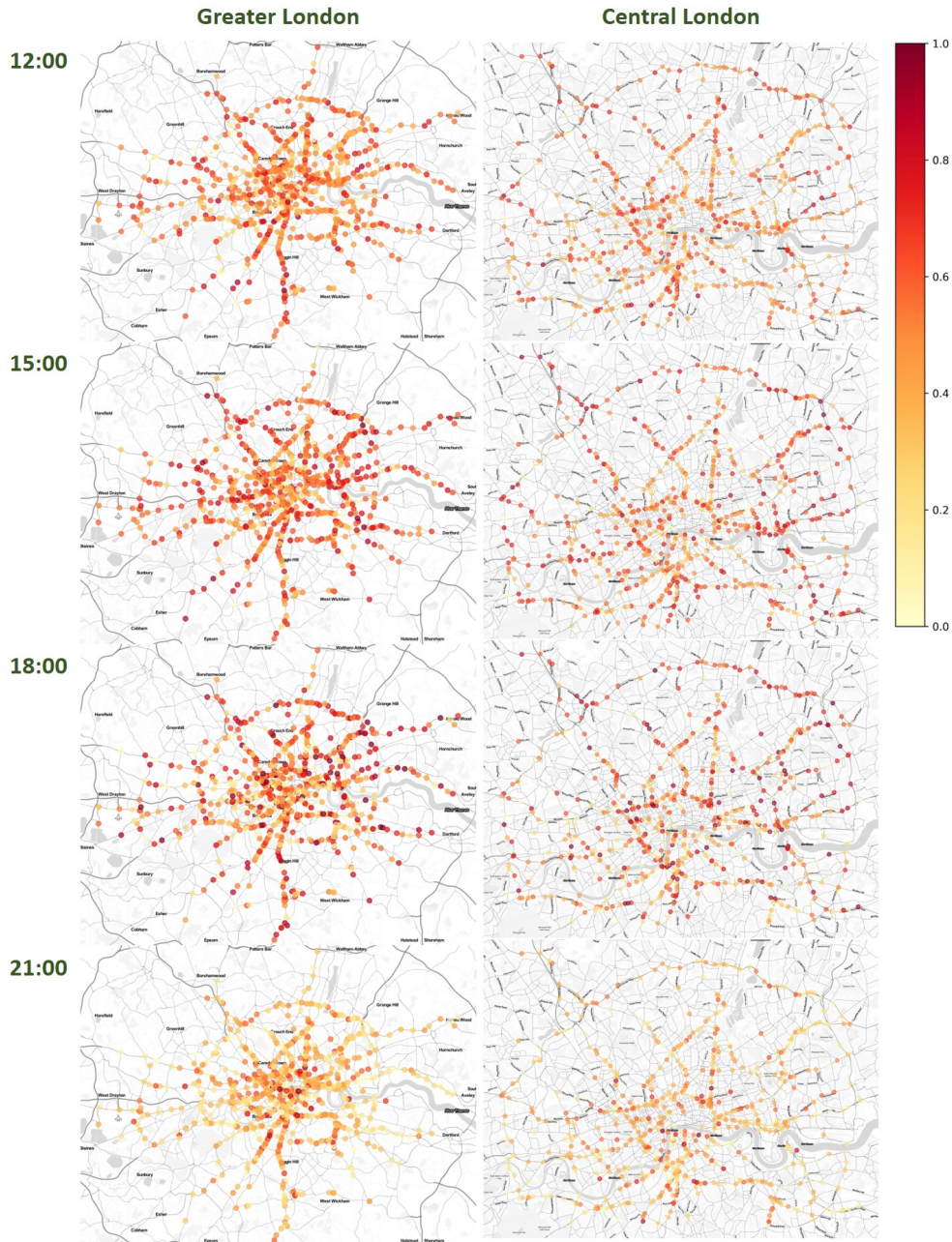


**Figure 6:** Normalised level of cars within the M25 (left) and inner city London (right) throughout the second half of Wednesday 31 July 2019. The M25 is the major motorway encircling Greater London

spreading is when roads cannot become more congested and drivers choose to travel at off-peak times, thus extending the times roads remain congested (Allen et al., 2016).

The raw count visualisation for car objects reveals a darker ring along the north circular highway. This is likely because JamCams along the highway capture much larger areas of roadway as opposed to cameras within cramped central London. This data may therefore be used to infer the vantage point of JamCam cameras, which may be useful for further tuning the detector on a vantage point height basis, or be used as additional context data about the JamCam.

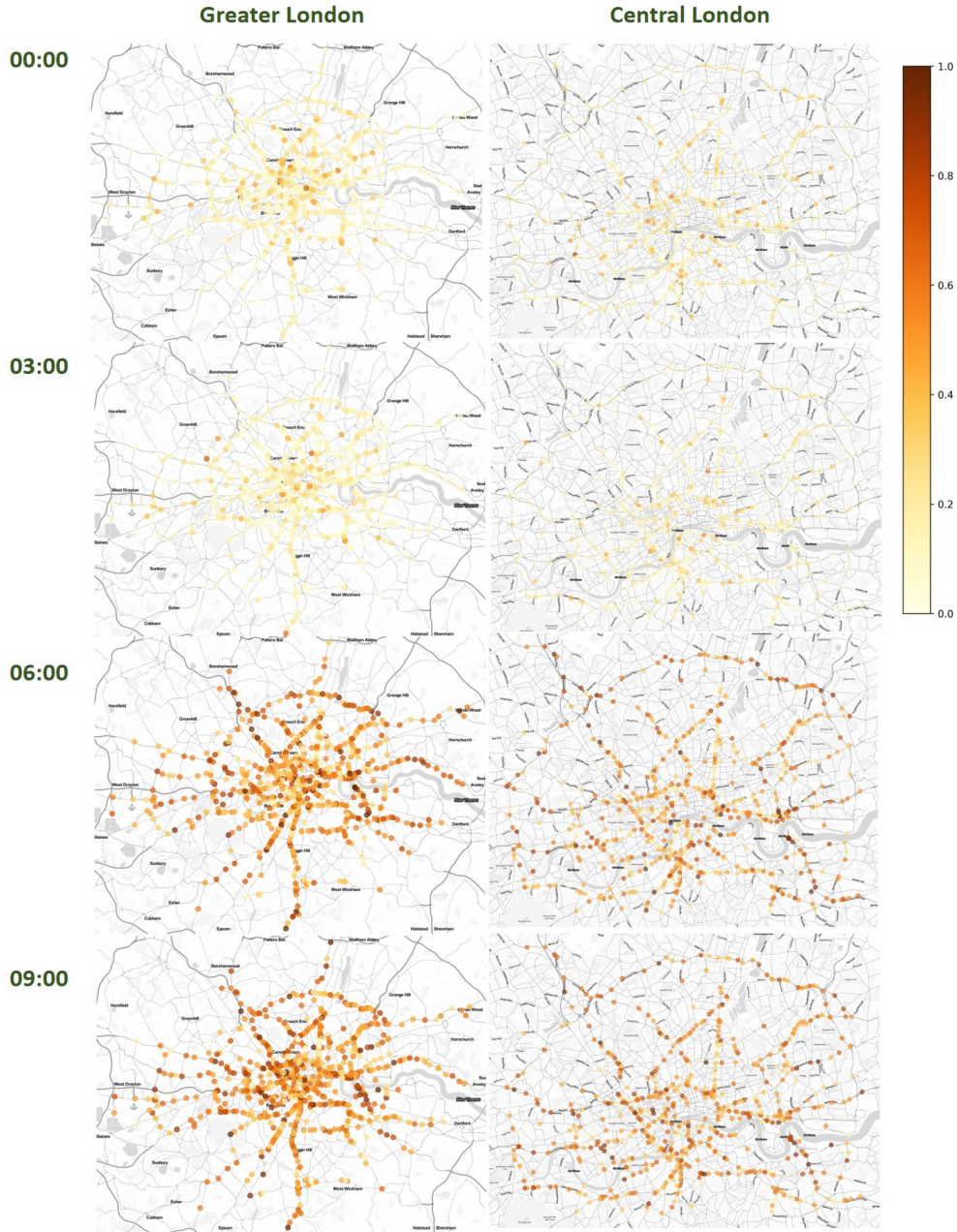


**Figure 7:** Normalised level of trucks within the M25 (left) and inner city London (right) throughout the first half of Wednesday 31 July 2019. The M25 is the major motorway encircling Greater London

## 5.2. Detecting Trucks

As explained in Section 4.1, the COCO dataset does not differentiate between heavy goods vehicles (HGV) and light goods vehicles (LGV), labelling both as *truck*. According to (Allen et al., 2016), HGV and LGV traffic make 17% of London's total traffic, while cars make up to 80%. Figures 7 and 8 shows the relative level of trucks detected (also on Wednesday 31 July 2019).

The normalised truck detection plot, at first glance, looks similar to that of the car detection. It follows a similar traffic pattern of increasing in intensity in the early day (as shown in Figure 7) and decreasing in the 2nd half (as shown in Figure 8). It also has a relatively even spread in intensity across the whole of London during the day. Comparing

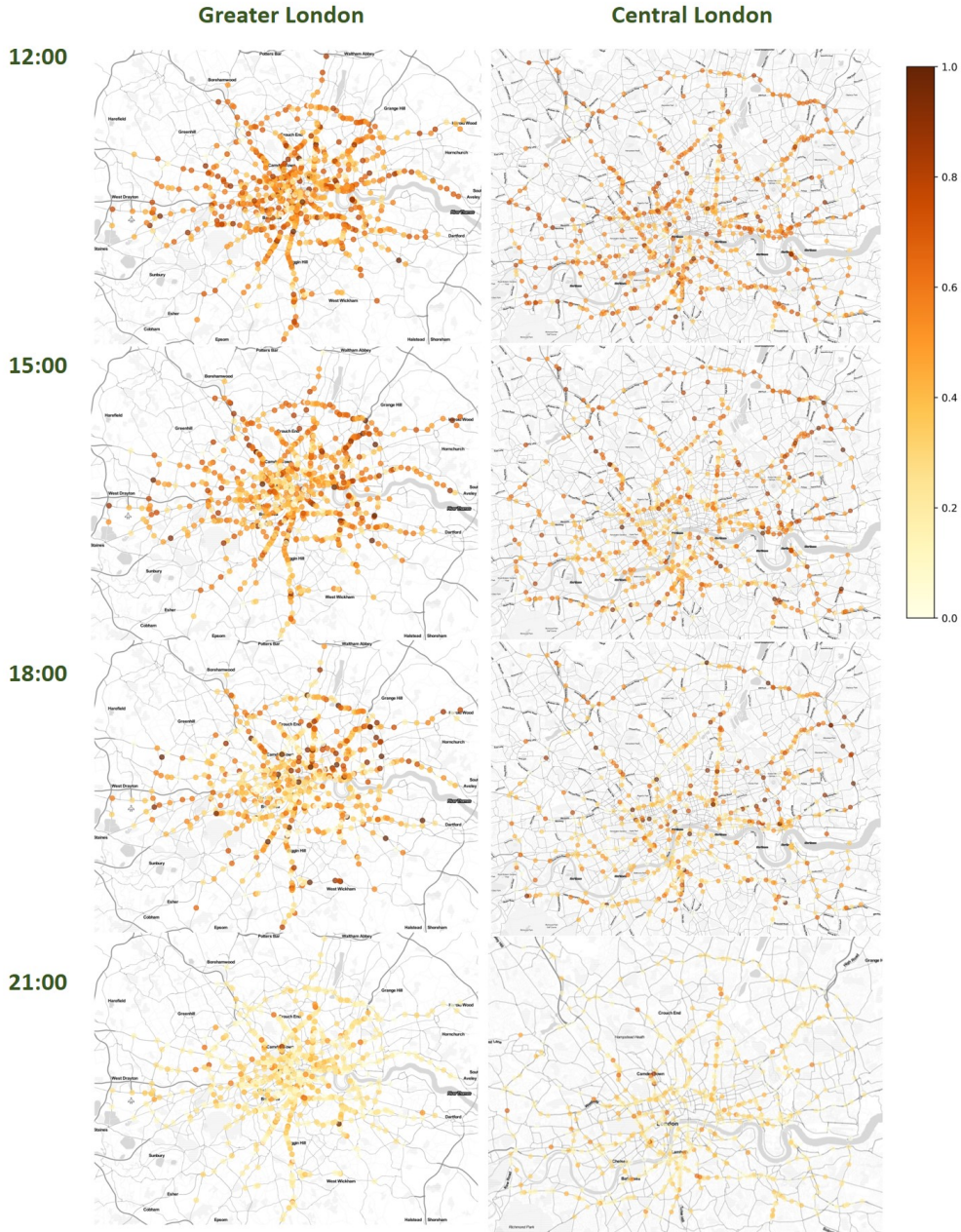


**Figure 8:** Normalised level of trucks within the M25 (left) and inner city London (right) throughout the second half of Wednesday 31 July 2019. The M25 is the major motorway encircling Greater London

it to the cycle for cars in Section 5.1, this cycle occurs earlier. The population of trucks increases significantly before cars do at 5am, and starts decreasing at 4pm. It is also observed that there is a less significant peak-spreading effect for trucks.

We also observed comparatively fewer truck detections on the weekends. This is consistent with the fact that most HGVs and LGVs are used for commercial purposes (Allen et al., 2016), and so significantly fewer trucks are expected on London's roads on Saturdays and Sundays.

Figures 7 and 8 also show that the spread of trucks is sparse enough to identify truck hotspots in central London. By applying a time series average on these datapoints, it could be possible to quickly identify areas where there is

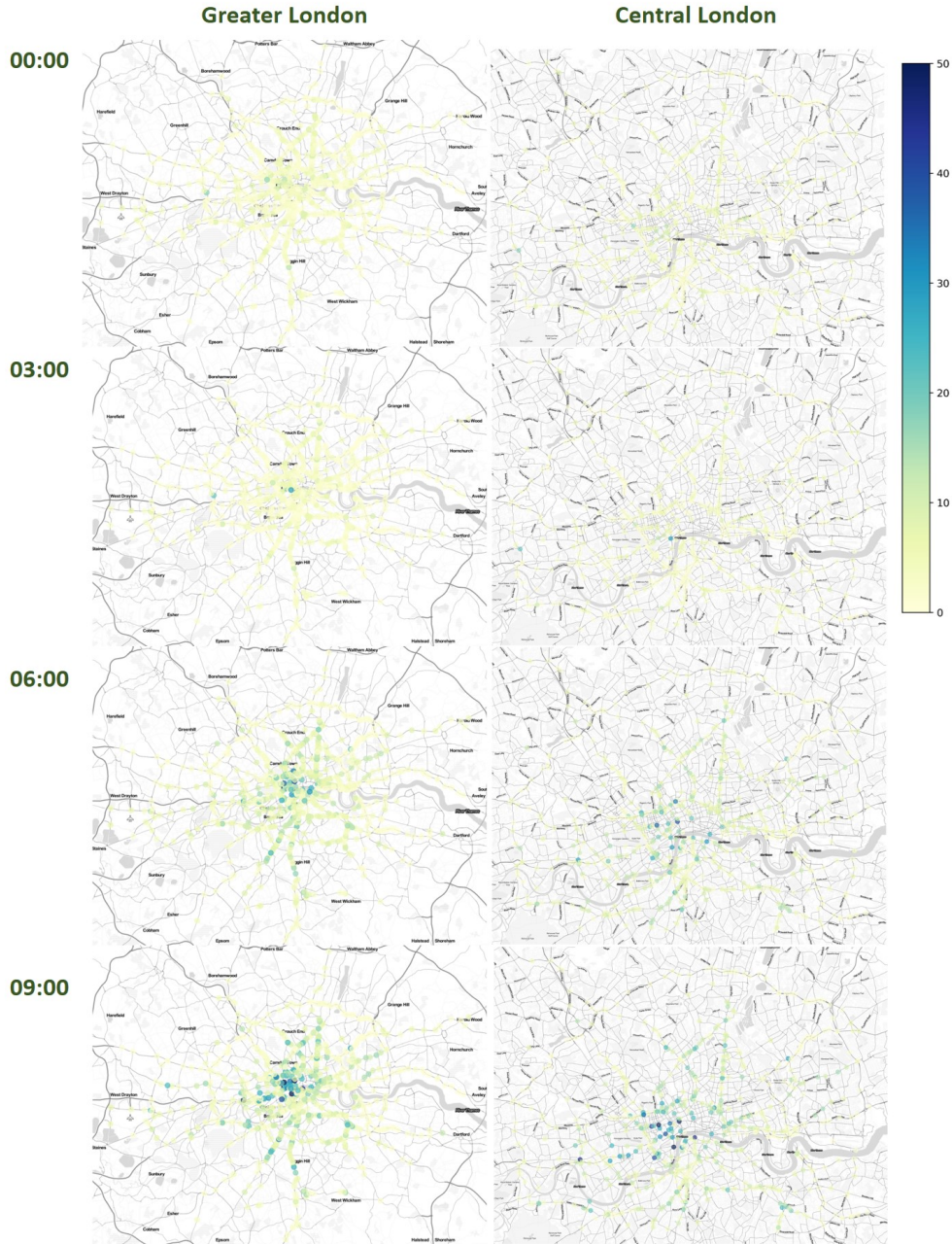


**Figure 9:** Average count of pedestrians within the M25 (left) and north & south circular (right) throughout the first half of Wednesday 31 July 2019. The M25 is the major motorway encircling Greater London

recent increase in HGV and LGV traffic.

### 5.3. Detecting pedestrians

Figures 9 and 10 show the raw count of pedestrians at different locations in London. Note that we used a different color scale to highlight that in this case values are not normalized (as they were in Figures 5–8 for cars and trucks) but the actual count (averaged in periods of 3 hours).

The raw plots demonstrate that pedestrians are mostly detected within central London; most likely because in this area, roads are narrow and pedestrian traffic is much higher. In contrast, most cameras outside of central London

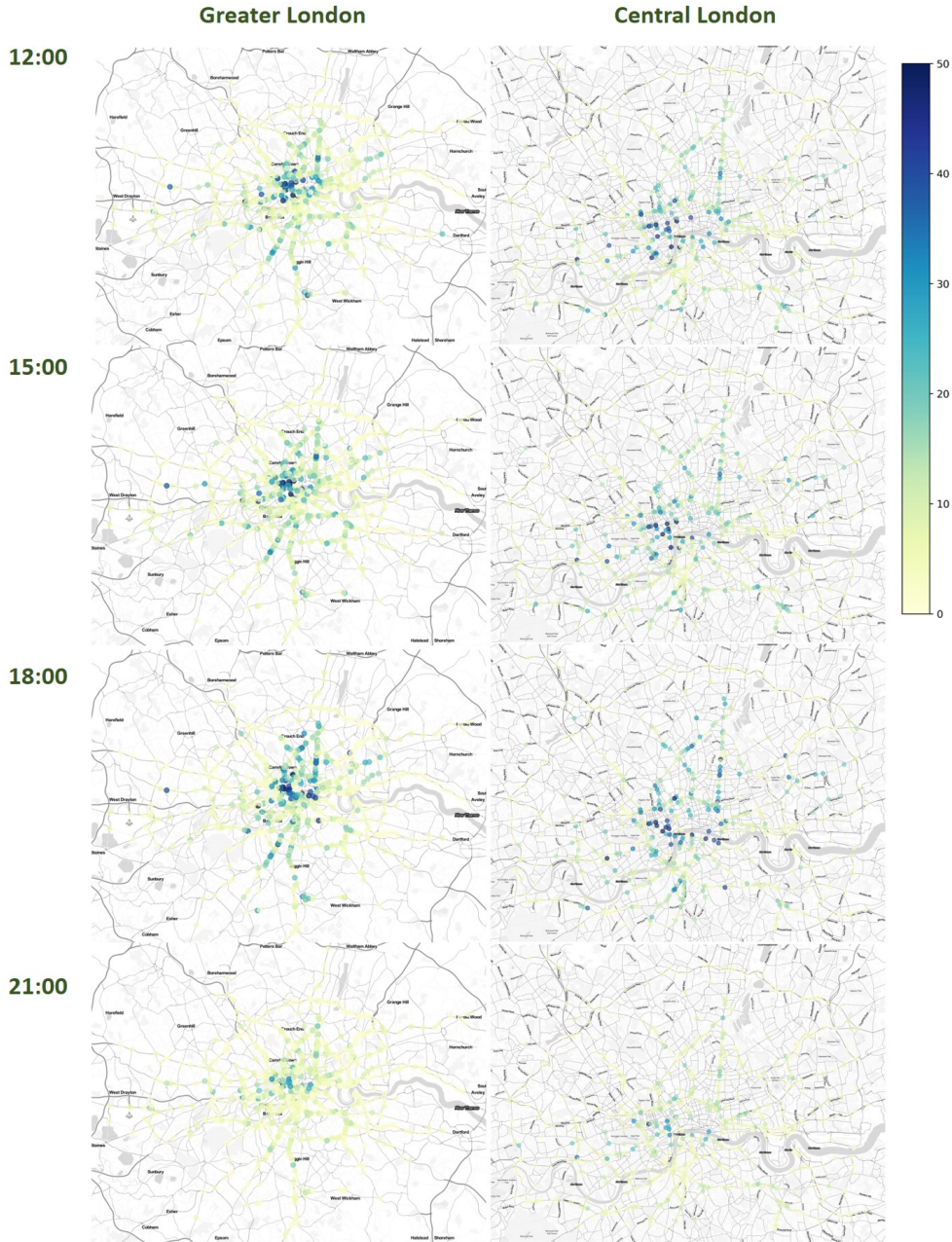
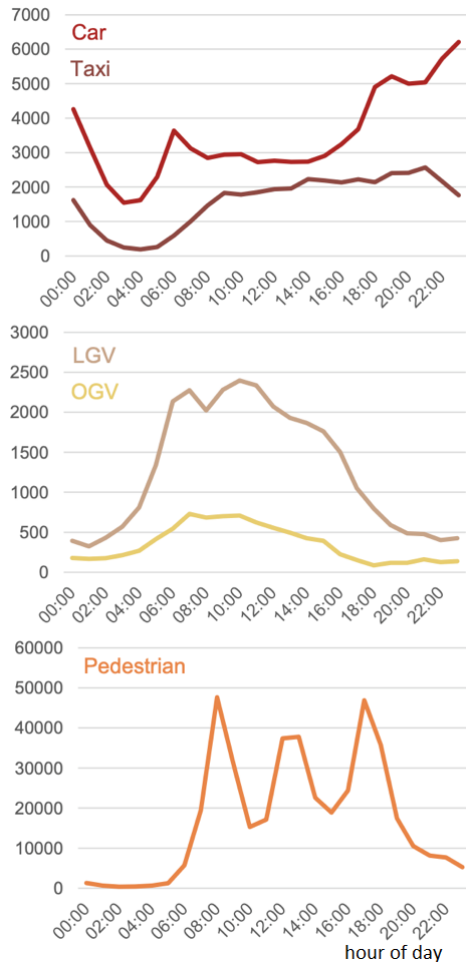


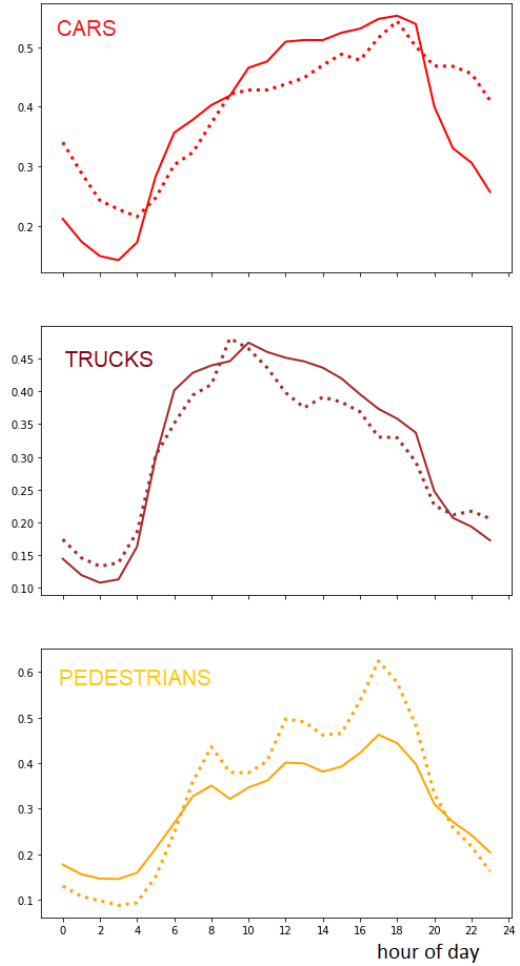
**Figure 10:** Average count of pedestrians within the M25 (left) and north & south circular (right) throughout the second half of Wednesday 31 July 2019. The M25 is the major motorway encircling Greater London

detect none to few pedestrian objects because they are mostly located along freeways. Therefore, meaningful pedestrian detections in London, through JamCams, are only available in zone 1 central London.

Particularly, the darkest spots in our plots are where cameras are positioned to view high levels of pedestrian traffic. These are *JamCam 7452* at Oxford Circus, *JamCam 6590* at Strand, *JamCam 7450* at Picadilly Circus, *JamCam 6501* at Parliament Square, *JamCam 6502* at Trafalgar Square and *JamCam 4601* at London Bridge. These cameras have higher vantage points as compared to other JamCams within inner city London. Naturally, with a broader field of view, they detect more objects—in some cases, more than 50 people during peak hours. On the whole, this suggests that our detection of pedestrian objects within inner city London is reasonably reliable despite the fact that TfL JamCams



(a) Average count of car, truck and pedestrian objects at each hour over three weeks in 2017, as reported by TCS (Strategic Transportation & Department of the Built Environment, 2018).



(b) Average levels of car, truck and pedestrian objects at each hour over three weeks in summer 2019, as measured by our detection system. Solid lines represent Greater London (within the M25), dotted lines represent inner London.

**Figure 11:** Comparing traffic trends for car, truck and pedestrian objects: (a) trends reported by TCS, (b) trends computed by our detection system.

are strategically placed to view road traffic rather than sidewalks.

With these findings in mind, the normalised plot versions of Figures 9 and 10 (not included in this paper) provide a reasonable illustration of pedestrian traffic flow. This however is limited to areas where JamCams can see larger portions of pedestrianised paths. A study on which cameras can serve as a reliable datapoint for pedestrians remains as future work. One method could be to use the raw count plot to quickly choose the cameras to ignore by applying a lower limit.

#### 5.4. Comparing against Traffic Composition Survey

The London Traffic Composition Survey (TCS) is carried out every 2 years by the city of London to provide details of the number and types of vehicles in London's streets. Data is gathered by humans posted at locations within the City of London borough (Strategic Transportation & Department of the Built Environment, 2018). Car and taxi objects, LGV and OGV (Ordinary Goods Vehicle), and pedestrians are the equivalent to car, truck and pedestrian objects respectively, as the build detection pipeline may not distinguish between an LGV or OGV, likewise a car or

taxi. Figure 11a shows the 24 hour time profiles of each object as reported in (Strategic Transportation & Department of the Built Environment, 2018).

Comparing these to their corresponding ones in Figure 11b, several general trends pointed out by the TCS holds true with the results from our detection pipeline. Figure 11b depicts the average level of car, truck and pedestrian objects as seen by our detection system over three weeks in the summer of 2019. The dotted line represents inner London and the solid line Greater London.

In the case of car objects, a steady rise in car traffic was observed by our detection pipeline from early morning until the evening. Around 10pm, however, the trends measured by our system and the TCS data diverge. The level of car objects in Greater London (dotted line), as measured by our system, start decreasing sharply around 7pm, yet according to TCS data, we should see it increasing. TCS reasons that the upwards trend in the graph (after 10pm) is due to the significant private car hire and taxi activity in the city. In fact, it can be seen that the dotted line, representing inner city cameras, aligns more closely to this trend. From the plot, it is clear that car activity in the late evening is more prevalent in the inner city where there is a busier 'night-time' economy. The Greater London cars' line aligns more closely with typical working commuter's activity, which makes sense as many cameras in the greater London region are positioned along freeways.

The shape of the sum of the LGV and OGV curves displays a very high-level similarity to the one generated by our detection system and described in Section 5.2. Both start rising sharply at 4am and peak between 8am and 12pm, before steadily dying down to pre-4am levels at 10pm. One can argue that this is not relevant, but it demonstrates that our system (only detecting objects in traffic cameras) is indeed able to capture the natural behavior of traffic flows throughout the day. For trucks, there is no big difference in trends between inner and greater London.

The pedestrian curve in Figure 11b also resembles the corresponding one in Figure 11a. Three distinct peaks are observable at 8am, 12pm and 5pm, which matches up with the typical daily behaviour of going to work, lunch hour, and evening commute home. As expected, the inner London graph (dotted line) exhibits more pronounced peaks and troughs, as inner London has more pedestrian traffic and several cameras here are positioned at pedestrian hotspots.

On the whole, the trends spotted by our detection pipeline (summarized in Figure 11b) closely align with those suggested by Figure 11a. The detection pipeline therefore provides an automated, real-time way of measuring traffic trends in London, comparable in accuracy to that of the TCS.

## 6. Conclusion

Automatic and accurate detection of traffic objects in video feed can be seen as a cost-effective solution to many cities in order to better understand mobility flows and to quickly react to slow traffic and other incidents.

The tooling described in this paper, open-sourced and publicly available, is a series of data processing pipelines to perform object detection on video feeds similar to London's TfL JamCam. Our solution makes use of the YOLOv3 model trained over the COCO dataset. It is fast enough to operate in near real time, being able to generate fresh detections within 5 minutes of a feed being released. It has also been custom-tuned using TfL JamCam footage to detect three of the most common objects in London's traffic scenes: cars, trucks and pedestrians. We tested the detection system by making it run continuously for 3 weeks, confirming that it can be used to reliably gather data over long periods of time.

The animations we have produced demonstrate how effective the system is at identifying pedestrian, truck and car objects in London. Trends observed in the produced data match up with that found in the London Traffic Composition Survey (TCS), proving it presents a sensible, real-time interpretation of the traffic situation in London.

Considering that cars, HGVs and LGVs compose 97% of London's total traffic (Allen et al., 2016), the obtained data on detections can very well be used as a gauge of London traffic levels. When the TfL JamCam webpage was launched, it was advertised as a way for the public to view live CCTV of JamCams to avoid congestion (BBC News, 2012). Our own experimentation proves that the process of determining congestion levels and traffic composition around different areas in London can be automated.

In our system, pedestrian detections are limited primarily to central London. With many debates about pedestrianising London's roads, automatically detecting crowd levels around London might be of interest. We were able to show that reasonable estimates can be obtained even with the existing orientation of cameras in London. Our choice of detector, YOLOv3, is however confirmed to perform poorly with crowded scenes. If the system is specifically used to detect people on newly pedestrianised roads, the results may be less reliable.

Overall, determining the level of trucks and cars provides more granular information about the traffic situation.

The ability to detect trucks can particularly be of use to estimating freight traffic patterns on different areas in London for congestion or pollution analysis. Detecting pedestrian, on the other hand, may be of more relevance with a growing movement to pedestrianise busy London streets.

In the current version, we are using a YOLOv3 model trained over the COCO dataset. This means that, in addition to identifying cars, trucks and pedestrians, we are getting identification of other objects like backpacks, handbags, hats, parking meters, fire hydrants, and stop signs (which are classes in the COCO dataset). An obvious further line of action would be to train YOLOv3 over a custom dataset of traffic images, focusing only on the objects of interest. For that, a labelled dataset that is not yet available would be needed.

All in all, we encourage transportation departments from cities around the world to adopt solutions such as the one we proposed in this paper and test them in their cities. Even without a specific fine-tuning of the detection models, the results will probably give a more than reasonable real-time picture of the traffic levels with very little effort.

## Acknowledgments

M. Molina-Solana was funded by the EU's H2020 programme (H2020 MSCA-IF g.a. No 743623 and No 754446).

## References

- Allen, J., Piecyk, M., & Piotrowska, M. (2016). *An analysis of road freight in London and Britain: traffic, activity and sustainability*. Technical Report University of Westminster. URL: [http://www.ftc2050.com/reports/westminster\\_road\\_traffic\\_final\\_Dec\\_2016.pdf](http://www.ftc2050.com/reports/westminster_road_traffic_final_Dec_2016.pdf).
- allroadcams (2020). Map of live traffic cameras in U.S. cities. URL: <http://www.allroadcams.com> accessed: 2020-02-18.
- BBC News (2012). Transport for London launches live traffic website. URL: <https://www.bbc.co.uk/news/uk-england-london-16779962> accessed: 2020-02-15.
- Chakraborty, P., Adu-Gyamfi, Y. O., Poddar, S., Ahsani, V., Sharma, A., & Sarkar, S. (2018). Traffic Congestion Detection from Camera Images using Deep Convolution Neural Networks. *Transportation Research Record: Journal of the Transportation Research Board*, 2672, 222–231. doi:10.1177/0361198118777631.
- Chin, A. T. (1996). Containing air pollution and traffic congestion: transport policy and the environment in Singapore. *Atmospheric Environment*, 30, 787–801. doi:10.1016/1352-2310(95)00173-5.
- Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object Detection via Region-Based Fully Convolutional Networks. In *Proc. 30th International Conference on Neural Information Processing Systems (NIPS'16)* (pp. 379–387). Curran Associates Inc. doi:10.5555/3157096.3157139.
- Deloitte (2017). *Assessing the value of TFL's open data and digital partnerships*. Technical Report Transport for London. URL: <http://content.tfl.gov.uk/deloitte-report-tfl-open-data.pdf>.
- Girshick, R. (2015). Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)* (pp. 1440–1448). Santiago, Chile. doi:10.1109/ICCV.2015.169.
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 580–587). doi:10.1109/CVPR.2014.81.
- He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*. doi:10.1109/ICCV.2017.322.
- Kabakus, A. T., & Kara, R. (2017). A performance evaluation of in-memory databases. *Journal of King Saud University - Computer and Information Sciences*, 29, 520–525. doi:10.1016/J.JKSUCI.2016.06.007.
- Li, X., Liu, Y., Zhao, Z., Zhang, Y., & He, L. (2018). A Deep Learning Approach of Vehicle Multitarget Detection from Traffic Video. *Journal of Advanced Transportation*, 2018, 1–11. doi:10.1155/2018/7075814.
- Lin, J.-P., & Sun, M.-T. (2018). A YOLO-Based Traffic Counting System. In *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)* (pp. 82–85). doi:10.1109/TAAI.2018.00027.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollar, P., & Zitnick, L. (2014). Microsoft COCO: Common Objects in Context. In *Proc. of the European Conference of Computer Vision (ECCV)* (pp. 740–755). European Conference on Computer Vision. URL: <https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/>. doi:10.1007/978-3-319-10602-1\_48.
- Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2020). Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, 128, 261–318. doi:10.1007/s11263-019-01247-4.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In *14th European Conference on Computer Vision (ECCV 2016)* (pp. 21–37). doi:https://doi.org/10.1007/978-3-319-46448-0\_2.
- London Councils (2020). About the London Lorry Control Scheme. URL: <https://www.londoncouncils.gov.uk/services/london-lorry-control/about-llcs> accessed: 2020-04-19.
- Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014.
- Olson, R., Calmels, J., Abecassis, F., & Rogers, P. (2016). NVIDIA Docker: GPU Server Application Deployment Made Easy. URL: <https://devblogs.nvidia.com/nvidia-docker-gpu-server-application-deployment-made-easy/>.
- Oltean, G., Florea, C., Orghidan, R., & Oltean, V. (2019). Towards Real Time Vehicle Counting using YOLO-Tiny and Fast Motion Estimation. In *2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME)* (pp. 240–243). doi:10.1109/SIITME47687.2019.8990708.

- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788). Las Vegas, NV, USA. URL: <http://pjreddie.com/yolo/>. doi:10.1109/CVPR.2016.91.
- Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement, . URL: <http://arxiv.org/abs/1804.02767>.
- Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39, 1137–1149. doi:10.1109/TPAMI.2016.2577031.
- Shi, H., Liu, Z., Fan, Y., Wang, X., & Huang, T. (2017). Effective object detection from traffic camera videos. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (pp. 1–5). IEEE. doi:10.1109/UIC-ATC.2017.8397671.
- Strategic Transportation, & Department of the Built Environment (2018). *Traffic in the City*. Technical Report City of London London. URL: <http://democracy.cityoflondon.gov.uk/documents/s91800/Appendix+1+-+Traffic+in+the+City+2018.pdf>.
- Tao, J., Wang, H., Zhang, X., Li, X., & Yang, H. (2017). An object detection system based on YOLO in traffic scene. In *Proc. 6th International Conference on Computer Science and Network Technology (ICCSNT)* (pp. 315–319). Dalian, China. doi:10.1109/ICCSNT.2017.8343709.
- Transport for London (2015). *Transport for London Syndication Developer Guidelines*. Technical Report. URL: <http://content.tfl.gov.uk/syndication-developer-guidelines.pdf>.
- Transport for London (2020a). Map of live traffic cameras in London. URL: <https://www.tfljamcams.net> accessed: 2020-02-18.
- Transport for London (2020b). Our open data - Transport for London. URL: <https://tfl.gov.uk/info-for/open-data-users/our-open-data#on-this-page-4>.
- Transport for London (2020c). Transport for London API. URL: <https://api-portal.tfl.gov.uk/>.
- Transport for NSW (2020). Live Traffic New South Wales. URL: <https://www.livetraffic.com> accessed: 2020-02-18.
- Tung, C., Kelleher, M. R., Schlueter, R. J., Xu, B., Lu, Y.-H., Thiruvathukal, G. K., Chen, Y.-K., & Lu, Y. (2019). Large-Scale Object Detection of Images from Network Cameras in Variable Ambient Lighting Conditions. In *Procs. 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. San Jose, CA, US. doi:10.1109/MIPR.2019.00080.
- Wang, P., Hunter, T., Bayen, A. M., Schechtner, K., & González, M. C. (2012). Understanding road usage patterns in urban areas. *Scientific reports*, 2, 1001.
- Wootton, J., García-Ortiz, A., & Amin, S. (1995). Intelligent transportation systems: A global perspective. *Mathematical and Computer Modelling*, 22, 259–268. doi:10.1016/0895-7177(95)00137-Q.
- Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 30, 3212–3232. doi:10.1109/TNNLS.2018.2876865.