# End-to-End Traffic Flow Rate Estimation From Compressed Video Streams

Benjamin Deguerre, Clement Chatelain, Gilles Gasso

# End-to-end traffic flow rate estimation from MPEG4 part-2 compressed video streams

Benjamin Deguerre, Clement Chatelain, Gilles Gasso

***Abstract*—Automatic traffic surveillance usually relies on the estimation of traffic flow parameters through either dedicated sensors or the processing of road surveillance cameras. However, dedicated sensors are expensive to deploy and maintain. Moreover, available video processing algorithms usually require a complex multi-step pipeline, unsuited for large scale deployment. Herein, we address the problem of automatically estimating the flow rate (number of vehicles/unit of time) from surveillance cameras at low computation cost. To do so, we rely on end-to-end deep architectures applied to compressed MPEG4 part-2 video streams issued from road surveillance cameras. By leveraging the approximate flow representation induced by the compression, we heavily reduce the computation and memory requirements. We propose three end-to-end deep architectures using this coarse pixel flow representation as input. We also release two datasets, one based on synthetic videos and one collected on industrial tunnel cameras. By training the deep models on the newly introduced datasets, we evidence the effectiveness of predicting the flow rate directly from MPEG4 part-2 compressed video streams. We demonstrate an improved accuracy in comparison with a more classical RGB-based architecture and show an impressive speed up of $\times 2065$ at prediction time.**

***Index Terms*—MPEG4-part2, Deep Learning, Compressed video, Motion Vectors.**

## I. Introduction

Traffic management and surveillance systems are key components to ensure the safety of road users. Such systems usually rely on a few fundamental traffic parameters (flow rate, density, velocity, etc.) to produce an overview of the road traffic [1]. Usually, these traffic parameters are acquired or estimated using induction loops. Induction loops are electromagnetic sensors embedded in roads that activate when a vehicle passes. Such sensors allow for a microscopic (per vehicle) analysis of the flow of vehicles and therefore provide a fine-grained analysis of the traffic flow in real-time. However, as they are recessed in the road, they are expensive to deploy and maintain: for each installation/maintenance operation, one needs to dig-in the road to access them. Because of these costs, induction loops are deployed at strategic areas on the road network, leaving large road sections without direct flow analysis.

While induction loops are scarcely deployed, cameras are extensively used for safety reasons: operators need to see the road in case of incidents (accidents, congestion, etc.). Therefore, the exploitation of these cameras for the estimation of traffic flow parameters would prove beneficial, allowing to cover a larger portion of roads while limiting additional deployment costs. With this in mind, many works have leveraged the recorded RGB videos to provide with real-time traffic flow estimation. Early approaches (e.g. [2], [3], [4]) mainly rely on handcrafted features and complex processings. Such methods are not suitable for large scale deployment as they usually need to be calibrated for each camera. More recently, thanks to the NVIDIA AI CITY challenge [5], [6], [7], [8] deep learning based methods have emerged (e.g. [9], [10], [11]). However, as the challenge focuses on microscopic traffic flow estimation, rather than macroscopic flow prediction, the proposed approaches usually rely on a classical, and computationally expensive, detect-and-track pipeline. Hence, these methods are also not suitable for large scale deployment.

Fig. 1. Visualization of a video stream frame alongside its compressed representation. From left to right are the original RGB frame, the residual image and the Motion Vector (MV) representation. Only the moving vehicles generate information to be compressed. Note that motion vectors point in the opposite direction of the vehicles flow as they refer to previous frames.

Traffic cameras, and more generally surveillance cameras, have the peculiarity that they record a fixed background, with only the objects of interest moving on screen. As video compression algorithms usually only encode differences between images, the compressed representation appears relevant for flow prediction as it extracts the moving objects *per design* (c.f. Figure 1). Therefore, many works (e.g. [12], [13], [14]) have tried to leverage the compressed video data to estimate various flow parameters. Still, these methods heavily rely on handcrafted features such as texture or displacement information, limiting *de facto* their large scale deployment. Overall, the lack of fully learned methods stems from the fact that large datasets for traffic flow analysis and estimation are scarce. Without access to induction loops, the annotation of such dataset requires to detect and track each vehicle throughout the video frames, and to extrapolate the traffic flow parameters from visual cues, such as lane marking. Hence, annotating hours of traffic videos reveals a daunting task.

In this work, we propose a new method for estimating the flow rate of objects moving on fixed background from

MPEG4 part-2 compressed videos (see Figure 2 for an illustration). Especially, we get rid of the pre-defined hand-crafted procedures and instead we introduce a deep learning approach that allows to learn the target task in a end-to-end manner. Specifically, we solely rely on motion vectors to efficiently estimate traffic flow rate through dedicated deep models. As learning deep architectures requires large amounts of labelled data, we introduce two new datasets: the Moving Digits (synthetic data) and the Urban Traffic Camera (UTCam) dataset (onsite data). Extensive evaluations on these datasets demonstrate impressive speed-up gains while improving the accuracy when compared with a classical detection-tracking-estimation model. The contributions of the paper can be summarized as follows:

- We cast the traffic flow rate estimation problem as a regression problem using the compressed MPEG4 part-2 videos as input.
- Deep yet lightweight and efficient architecture are proposed to directly handle these compressed data.
- We introduce two datasets, Moving Digits, based on generated videos[1], and UTCam, composed of 58 hours of videos recordings from multiple tunnel cameras spanning over two days[2].

The remainder of the article is divided as follows. We review related works in section II. We formulate the problem and detail the proposed methods in section III. We introduce the new datasets in section IV. And, in section V we detail the experiments and results.

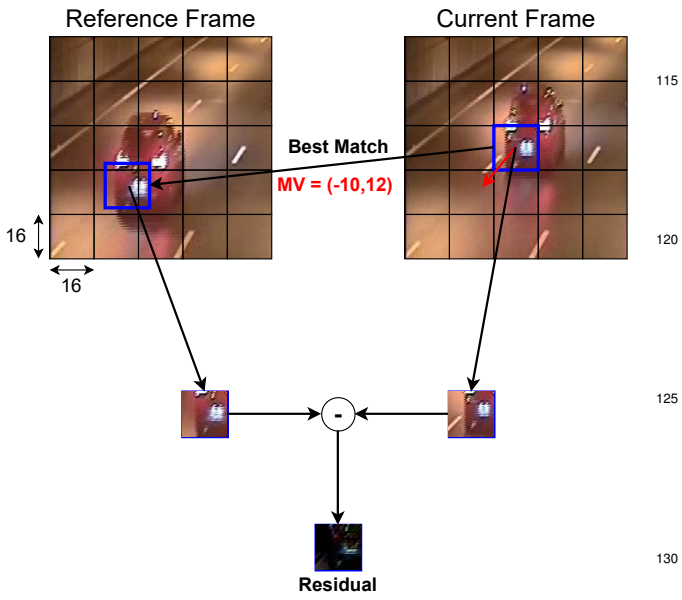



Fig. 2. The MPEG4 part-2 compression algorithm is based on *block-wise* motion compensation [15]. It introduces two main compression artifacts: Motion Vectors (MVs) and residuals. MVs encode the displacement of each block in the current frame relative to its best match in the reference frame[5]. Residuals are the difference information between two matched blocks. As shown, MVs are vectors of size 2 while a residual block is of size 16 x 16 pixels[6].

[1]The code for the dataset generation is available at the following repository.
[2]The dataset can be downloaded at the following link.

## II. Related works

Available methods for traffic flow estimation can be divided into two main groups: tracking-based and feature-based methods. While the former bases its estimation on the computation of vehicles tracks, the latter directly estimates the flow parameters from salient features. Hereafter, we describe both categories. Note that, we restrict the reviews to video-based methods, knowing that multiple other ways to predict traffic flow parameters exist (GPS data [16][17], traffic sensor data [18] [19], etc.). Such methods are out of our scope.

### A. Tracking-based estimation

In tracking-based estimation each vehicle is detected and tracked and then, the flow parameters can be naturally estimated.

Early methods are based on hand-crafted pipeline to carry the tracking and estimation. They are aimed towards the surveillance of roads from Unmanned Aerial Vehicles (UAVs) [20], [21] or from high point of views [4]. This set-up has the main advantage to avoid vehicles occlusions thanks to the position of the camera. For instance, in [20] the authors carry vehicles counting through a multi-step procedure based on Shi-Tomasi features [22], a Kanade-Lucas-Tomasi (KLT) feature tracker [23], [24] and clustering. [21] relies on Haar cascade, a classification network and a KLT tracker to estimate flow density and speed. Similarly, [4] detects and tracks vehicles from high view points by relying on a Lucas-Kanade (LK) tracker [23]. Few works have also tackled the flow estimation problem in the context of pedestrian surveillance. For instance, [25] carries foreground segmentation and use optical flow information to count the pedestrians crossing a predefined line.

Recently, newer methods based on deep learning have emerged [9], [10], [11], [26], [27], [28], [29], [30], [31]. However, due to the scarcity of data, they are mostly related to the yearly NVIDIA AI city challenge [5], [6], [7], [8]. As the challenge evaluation is based on vehicle's microscopic velocity estimation, most of these methods hinge on closely related steps. They apply the same global scheme of a deep detector followed by multi-object tracking, 2D to 3D projection and flow parameters estimation. Mainly, these methods differ in few steps in the processing pipeline. For instance, the detection network is either: Faster R-CNN [9], [27], [28], Mask R-CNN [26], [11], [29], [30] or YOLOs [10], [31]. The tracking algorithm can variate: median flow [11], hungarian matching [30] or DeepSORT [31], [27]. Also, the projection-estimation methods can differ: areas from landmarks [9], vanishing point estimation [11], [26] or intersections crossing [27], [29], [30]. And, unrelated to the AI City Challenge, [32] uses a Faster R-CNN to detect vehicles from an UAV and then estimates traffic flow parameters based on manually computed lane lengths.

Finally, let mention that few research works have proposed to detect objects using compressed videos. For instance, [33] extracts and clusters groups of MVs to compute handcrafted features that are then provided to a gaussian radial basis

[5]Note that herein, the reference frame is always the previous frame.
[6]The 16 x 16 block could actually be sub-divided depending on the compression parameters. This is not the case for this work.

function network to classify the level of service (free flow, congested, etc.). Also leveraging MVs to carry object detection, [34] adds RGB information to avoid mixing up multiple vehicles which can then be tracked and counted.

Overall, the tracking-based framework has been improved in the recent years. Indeed, the NVIDIA CITY challenge [5], [6], [7], [8] allowed the replacement of old handcrafted detectors with newer learned detectors. However, even the newer methods still heavily rely on heuristics for the estimation of the various flow parameters. Furthermore as they are based on deep RGB-based image detectors, scaling up such methods would prove computationally expensive.

### B. Estimation from salient features

Another trend of methods directly extracts salient features to predict traffic flow parameters. A large body of such methods is based on the processing of RGB-based inputs. For instance, [2] and [3] lie on the usage of intensity profiles to determine the speed of vehicles. The main idea is to segment road lanes with the presence or absence of vehicles and to compute the shift between two segmentations at consecutive time steps. Taking another approach, [35] extracts features such as edges or textures to predict density through regression. And, relying on the computation of a mean background image, [36] extracts the changes in new frames and then estimates the traffic flow from the ratio of foreground over background pixels.

Although RGB-based methods are efficient, they do not leverage the flow information encoded in videos by the compression algorithms. In particular, the MV frames and the residual images, that respectively encompass flow and texture information, can be used. For instance, [12] and [13] extract the MVs from the video flux and filter them based on their intensity and the texture information of the DCT coefficients. Then, they project them from the 2D frame space into the 3D road space to estimate the speed and density of vehicles on the road. Also using both MV and residual information, [14] handcrafts a multi-dimensional feature vector (based on variables such as MV mean, standard deviation, etc.) which is fed to Gaussian Mixture Hidden Markov Models to estimate the traffic state (stopped, empty, etc.). Ignoring part of the compressed information, other methods solely rely on the MVs to estimate the traffic flow parameters. For instance, [37] uses the MVs for calibration of the camera (detection of the area to process in the frame) and to estimate the mean velocity. [38] extracts various features (area, perimeter, shape of MV blobs, histogram of MVs, etc.) and then estimates the number of vehicles using a regression model. Finally, [39] leverages the MVs to correctly position a controllable camera and to estimate the traffic flow parameters in the RGB domain.

As for the tracking-based methods, the estimation of traffic flow parameters from extracted features does not scale to large road networks because of the required manual calibration steps.

## III. Problem Statement and Proposed Solution

Let focus on our task of interest, the estimation of the flow rate from surveillance cameras. In particular, we address the problem from a macroscopic perspective. We first formalize the learning problem and then, we detail the proposed solution as well as a baseline method.

### A. Problem formulation

We abstract from the vehicle counting to a more general object related formulation. Let $S$ be a set of training examples drawn from a distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Q}}$. The input space $\mathcal{X} \in (\mathbb{R}^{H\times W\times C})*$ is the set of all possible sequences (a sequence is denoted by the $*$ symbol) of video frames (compressed or not) of height $H$, width $W$ and with $C$ channels. The output space $\mathcal{Q} \in \mathbb{R}_+$ is the set of flow rate values $q$, defined as the number of objects $\Delta N$ that have crossed a given image section during time interval $\Delta T$. Note that we consider the average value in case of multiple object flows. Each example in $S$ consists of a pair $(\mathbf{x}, q)$. In general, the sequence $\mathbf{x}$ is of length $L$ (in frames), over which the value $q$ is computed. We also define $l \in [1, L]$ as the frame index of a given input sequence. As $q$ refers to the average number of passing objects over the $L$ frames, we make the assumption that each sequence is statistically independent from the others. The aim is to build a regression model $h$ such that:

$$q = h(x_L, \ldots, x_l, \ldots, x_1) + \epsilon = h(\mathbf{x}) + \epsilon, \qquad (1)$$

where $\epsilon$ is the measurement error. Hereafter, we will design $h$ as a deep network trainable in end-to-end manner. It is important to note that $h$ must not require high computation and bandwidth resources as we aim to produce a highly scalable solution.

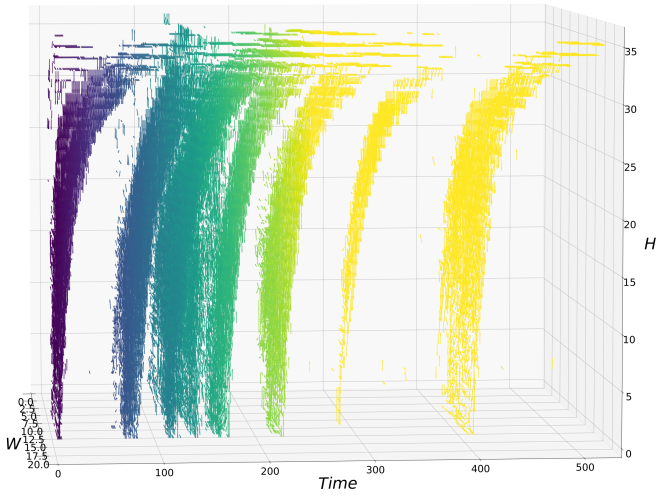



Fig. 3. 3D plot representation of Motion Vectors for vehicle flux within a tunnel. The plot shows 9 vehicles passing over 3 different lanes.

### B. Proposed solution: learning from motion vectors

Equation 1 addresses the counting problem by analyzing each frame within the targeted time interval. Naively using the plain RGB videos (see subsection III-C) will lead to cumbersome deep networks not suited to a scaled deployment. Rather, we seek to leverage the compressed video streams in

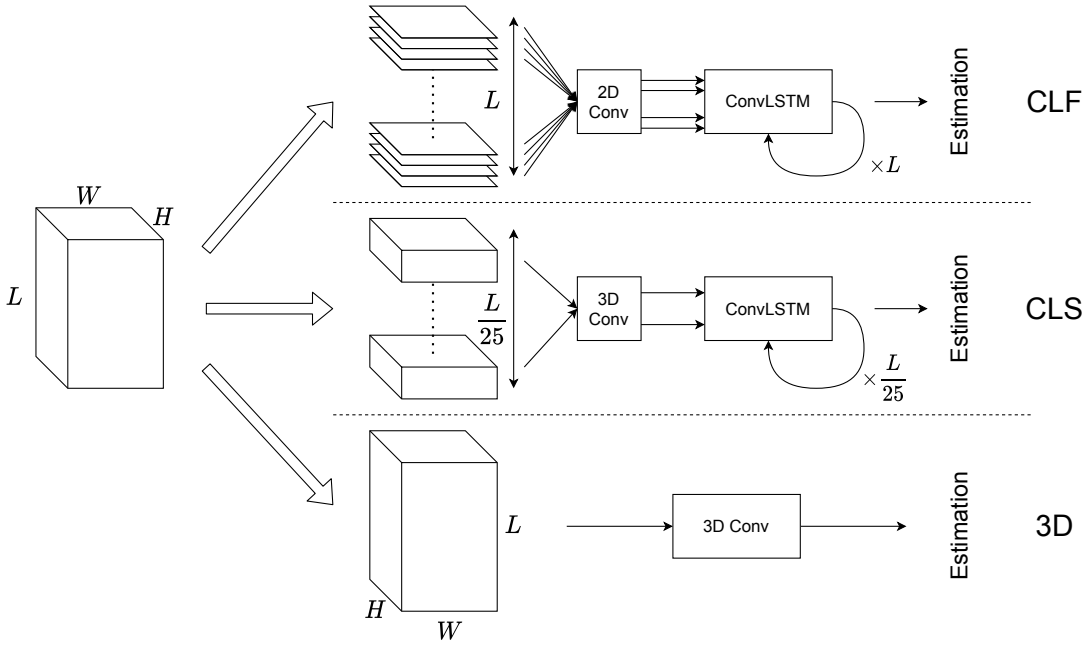



Fig. 4. The three proposed approaches for the estimation of the flow rate through regression. (Top) the *DeepMotion CLF* approach, (middle) *DeepMotion CLS* and (bottom) *DeepMotion 3D*.

order to benefit from a more compact representation as input. The MPEG4 part-2 norm uses Residual frames and MV frames to compress videos (see Figure 2 for a reminder of the MPEG4 part-2 compression). Residual frames contain the difference between video frames. Although they are sparse due to the removal of the background, residual frames are identical to RGB frames, both in shape and type of data (textures). As such, using Residual frames, in lieu of RGB frames, would not resolve the computation and memory issues. Conversely, MV frames contain a coarse representation of the flow of pixels between frames. They compress the information by providing the motion information for blocks of $16 \times 16$ pixels. Thus, MV frames allow to comparatively reduce the dimensionality of the input by a factor 256. This motivates us to base our proposed networks solely on sequences of MV frames. We now discuss different approaches to handle sequential and spatial aspects of the modelling.

*1) Sequential modelling:* We aim to devise models that can process MV frames to predict the associated $q$ values. Such processing requires to jointly analyse spatial and temporal information. Figure 3 illustrates the spatial and temporal context for a given sequence of MV frames. We first address the problem through sequential modeling, decoupling spatial and temporal processing. As all MV frames are expected to have similar properties, the spatial processing can be addressed using shared 2D Convolution layers. As for the temporal processing, such problematic is often addressed through Recurrent Neural Networks (RNNs) [40], [41]. To account for spatial information, we consider recurrent network with ConvLSTM layers [42]. Therefore, the first proposed architecture is based on the combination of shared 2D convolutions and ConvLSTM. We call this network *DeepMotion CLF* (*DM CLF*) for DeepMotion ConvLstm Frame. The overall pipeline is shown in Figure 4, top panel.

*2) Sequential modeling through coarse spatiotemporal clustering: DM CLF* architecture may face the issue of modelling a long temporal dependency as the input sequence typically covers 20 seconds (about 500 frames). Moreover, RNN-based networks for long sequences do not allow to properly leverage the parallelization capabilities of the GPU. In order to circumvent these problems, we propose to process the input data second by second rather than frame by frame. By doing so, the ConvLSTM layer has fewer time steps to process, consequently limiting the effects of both lack of GPU parallelization and long-term dependencies. For this approach we keep the ConvLSTM layer for temporal processing. However, as the unit component is now formed of multiple consecutive frames (25 in our case), we replace the 2D convolutions by 3D convolutions. The resulting architecture is denoted as *DeepMotion CLS* (*DM CLS*) for DeepMotion ConvLstm Second. The model is detailed in Figure 4, middle panel.

*3) Full spatiotemporal modeling:* The third architecture considers the input sequence $\mathbf{x}$ as a whole and process the $L$ frames altogether through stacked 3D convolutions. As such, this architecture can be seen as the limit case of *DM CLS* where the frames grouping is carried over 20 seconds. Such approach maximizes the use of the GPU by fully leveraging its parallelization capabilities. We name this approach *DeepMotion 3D* (*DM 3D*). It is detailed in Figure 4, bottom panel.

To train the proposed networks, we want to minimize large prediction errors which could raise false alarms in surveillance systems. Therefore, we choose to use the Mean Squared Error (MSE) so as to more strongly penalize large errors:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(q_i - \hat{q}_i)^2, \tag{2}$$

where $n$ is the number of sample in the training batch and $q_i$ and $\hat{q}_i$ are respectively the annotated and predicted $q$ values.

### C. RGB Baseline: Detect and Track

We design a baseline RGB method that will serve as reference. It is based on the classical detection-tracking-estimation pipeline. In such case, the regression model $h$ consists in a three-step pipeline: 1) process each frame for detection, 2) generate tracks and 3) count the number of valid tracks to estimate $q$.

For detection, as the sought model is intended to be fast and to have low memory consumption we select the well known SSD detector [43] that trades off performances and speed.

Regarding the tracking system, various solutions exist, the most renown being Sort [44] (a combination of a Kalman Filter [45] for object displacement estimation and the hungarian algorithm [46] for object association between frames) and its recent deep variant DeepSORT [47]. Because the model is to be deployed on cameras that may have different setups (angle, distance to road, etc.), we choose to solely rely on the hungarian algorithm, with euclidean distance, so as to minimize the required manual calibrations.

Finally, for the estimation we manually set a virtual induction loop and count the number of detected tracks crossing the loop to estimate $q$. The overall procedure is detailed in Figure 5.

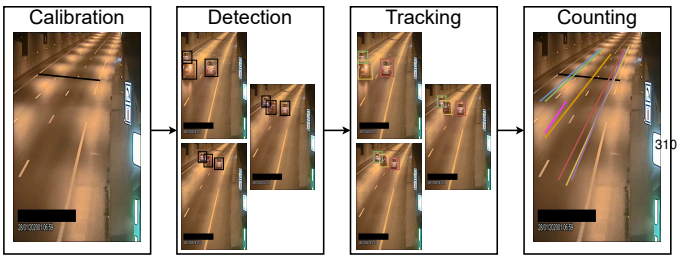



Fig. 5. Illustration of the RGB detect-and-track baseline method. First, the camera is manually calibrated by drawing a reference line (black line on the left image). Then, each frame is processed by the detector, leading to tracks computed using the hungarian algorithm. Finally, the tracks are used to compute the flow rate.

## IV. Datasets

To address the problem of traffic flow estimation we constitute and release both a synthetic and a real world datasets. The latter is constructed composed of collected video recordings and induction loop readings. Below, we introduce both datasets: Moving Digits and Urban Tunnel Cameras (UTCam). The characteristics of each dataset are summarized in Table I.

### A. A synthetic dataset: Moving Digits

To evaluate the proposed networks in various scenarios that will mimic real life conditions of road tunnel recordings, we design a synthetic dataset with controllable features. In particular, we target the two following issues: change of camera angle and change of number of lanes.

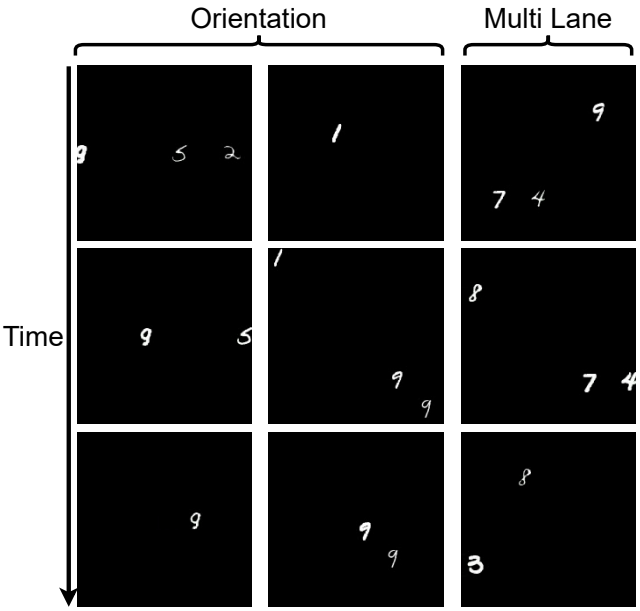



Fig. 6. Examples of simulated frames by our Moving Digits generator. The frames are shown in chronological order on each column. The first two columns show digits moving with different orientations. The last column illustrates digits moving on two separate flows. Same figures with different styles can follow each other, similar to cars of the same brand/colour in real life.

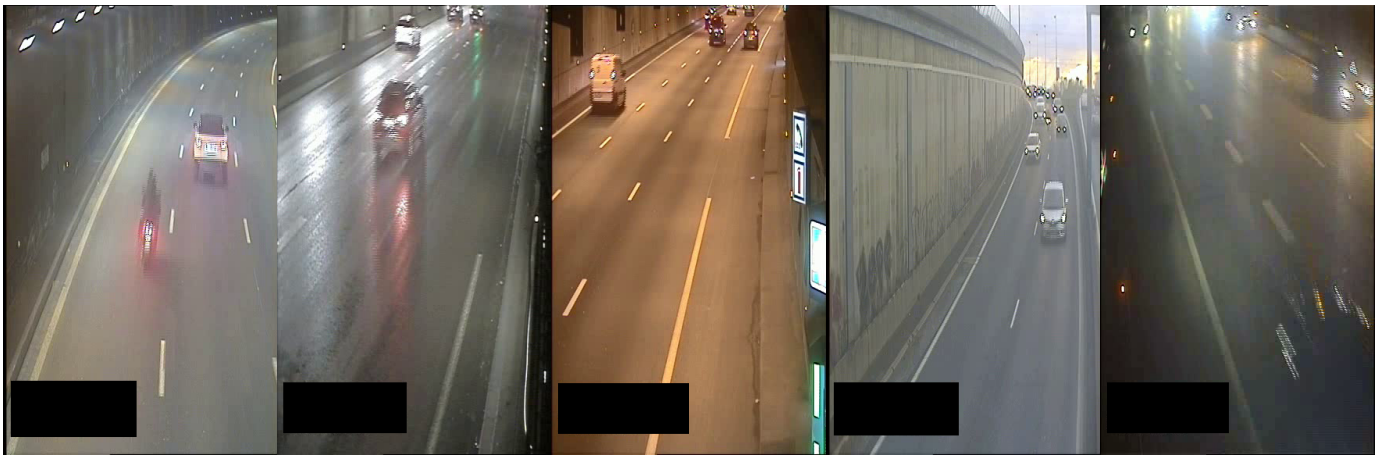
Moving Digits dataset is based on MNIST [48]. It is composed of randomly selected digits crossing a black screen in a given direction. We build a generator so as to control various parameters during generation. These parameters include:

- coordinates on the screen of the start and exit points (orientation),
- a scale factor (of the object size),
- the speed (number of frames for object to go from the starting point to the exit point),
- the maximum number of digits in the stream,
- the generation frequency ratio (probability of a new object being generated).

We generate multiple datasets covering the two mentioned issues (camera orientation and number of lanes). For the camera orientation, we generate datasets with a flow of digits that always cross the center of the video frames. We consider the following angle values $\{0, 45, 90, 135, 180, 225, 270, 315\}$, w.r.t the abscissa, $0°$ being a horizontal flux from left to right. Flows ending in the top of the frames ($45°$, $90°$, $135°$) can be considered to mimic tunnel cameras looking at the rear of vehicles ("Going"), the flows ending at the bottom ($225°$, $270°$, $315°$) mimic cameras looking at the front of vehicles ("Coming") and the remaining orientations ($0°$, $180°$) are in-between position neither "coming" nor "going". Finally, we generate videos with two flows (resp. above and below the horizontal axis) for the angle fixed at $0°$. For all those setups, we set the generation frequency ratio to 0.01, the maximum number of digits to 20 and the speed to 120 frames. MPEG4 part-2 encoded generated videos are 20 seconds long, at a frame rate of 25 FPS and 200x200 pixels in width and height. For each explored parameter, we simulate three sets (train,

TABLE I
SUMMARY OF THE SPECIFICITIES OF EACH DATASET

| Dataset | Number of datapoints | Number of configurations | Video | | | |
|---|---|---|---|---|---|---|
| | | | Size | Duration | FPS | Overlap |
| Moving Digits | 17,000 per configuration | 9 (8 orientations & 1 multi-flow) | 200x200 | 20s | 25 | NA |
| UTCam | $\approx 10,000$ | 5 cameras | 352x576 | 21s | 25 | 1s |



(a) $G1_3$ (b) $G2_4$ (c) $G3_3$ (d) $C1_2$ (e) $C2_4$

Fig. 7. Snapshots of the recorded cameras. Each camera is identified by a tag (G stands for Going and C stands for Coming). The index indicates the number of lanes. Black patches are used to anonymize the cameras.

validation and test) with respectively 10,000, 2,000 and 5,000 videos. Example of generated frames are shown in Figure 6. Note that, we apply random dilation and erosion onto the digits throughout their displacement to add some noise and avoid a too simplistic scenario.

### B. Real world Data

Annotating videos with flow information by hand is a complex task. We couple videos from Paris' tunnel roads with induction loop measurements to provide pairs of samples $(\mathbf{x}_i, q_i)$. Details of the annotation process are provided in the appendices, section A. The video recordings were carried over 2 days at various hours over 5 cameras. Twelve hours of videos were recorded per camera for a total of 58 hours (two hours were removed because of roadworks). This amounts to about 10000 datapoints. The resolution of the cameras is of 352x576 pixels. Each sample covers 21 seconds of recording, with an overlap (about 1 second) with previous and subsequent datapoints. For each of the datapoint, we collect the associated flow rate $q$ value, computed over 20 seconds non overlapping window, as well as the RGB, MV and residual frames[7].

Out of the five recorded cameras, three look at the back of the cars and two look at them approaching. We identify the two cameras looking at the vehicles approaching as C1-2 (Coming 1-2) and the three cameras looking at the rear of the vehicles as G1-3 (Going 1-3). As the number of lanes are not identical between the cameras, they are indicated as indexes to the camera IDs (see Table II). Finally, one camera ($C1_2$) is at the entry of a tunnel and is therefore subject to night and day illuminations. The specificities of the cameras are detailed in Table II. Snapshots of the cameras are shown in Figure 7. For each of the five cameras, $G1_3$, $G2_4$ and $G3_3$, $C1_2$, $C2_4$, we use the first day for training and validation (respectively 3,982 and 995 datapoints, or about 800 and 200 per camera) and the second day for testing (5,233 datapoints in total, or about 1,050 datapoint per camera).

[7]The residuals were not directly extracted from the video flux and are in the RGB space, not in their frequency representation.

TABLE II
DETAILS FOR EACH OF THE AVAILABLE CAMERAS

| Camera ID | Orientation | | Number of lanes | Is outside |
|---|---|---|---|---|
| | Coming In | Going | | |
| $G1_3$ | | X | 3 | |
| $G2_4$ | | X | 4 | |
| $G3_3$ | | X | 3 | |
| $C1_2$ | X | | 2 | Yes |
| $C2_4$ | X | | 4 | |

## V. EXPERIMENTAL EVALUATIONS

Experiments are conducted to assess the accuracy of the proposed deep architectures. First we investigate the strengths and limitations of the proposed methods in a controlled environment using the synthetic Moving Digit dataset. Then, we evaluate these deep models on the UTCam dataset and the results are compared with the baseline RGB detection method.

### A. Networks architectures

The tested networks follow the three architectures proposed in subsection III-B (*DeepMotion CLF*, *DeepMotion CLS*, *DeepMotion 3D*). Note that because the two datasets do not

have the same frame resolution, we use slightly different versions of the networks to fit the varying input frames dimensionality. Mainly, filters are added to the networks used for the UTCam dataset to account for the increase in the frame dimensions. All the architectures are available in the provided github repository.

For the *DeepMotion CLF* architecture, we use two 2D convolutions with stride 2 so as to reduce the spatial dimension of the input data. Then, we apply a ConvLSTM layer and finally use two consecutive dense layers. For the *DeepMotion CLS* network, we use two 3D convolutions, with stride 2. Further, we add an average pooling layer to reduce the dimensionality on the time axis. We also apply the ConvLSTM layer followed by dense layers. Finally, for the *DeepMotion 3D* architecture, we use two 3D convolutions followed by an average pooling layer and directly use two dense layers for prediction.

For the baseline RGB-based model, we use a SSD [43] network trained on a detection dataset extracted from the tunnel's cameras. The threshold for the selection of the detected boxes is set to 0.5. The virtual induction loops are manually set for each camera.

### B. Training and evaluation details

The networks are trained on a single NVIDIA GTX 1080 GPU. We use an Adam optimizer, no weight decay is applied and the batch size is set to 32.

Accuracy of the networks is evaluated based on the Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=0}^{N}|y_i - \hat{y_i}|, \tag{3}$$

as well as on the correlation coefficient $R$ between the predicted and real values:

$$R_{Y,\hat{Y}} = \frac{\mathbb{E}[(Y-\mu_Y)(\hat{Y}-\mu_{\hat{Y}})]}{\sigma_Y \sigma_{\hat{Y}}}. \tag{4}$$

These two metrics account for both the overall precision of the networks (MAE) as well as their capacity to follow the traffic trend ($R$). For each experiment, the networks are trained five times and the results are averaged.

Regarding inference speed, we evaluate the number of video streams that can be processed by a network in parallel (RealTime Stream Processing [RTSP]). We set the batch size to 8 and run 1000 predictions. As the proposed architectures are extremely fast due to their reduced size when compared with classical RGB-based networks, we pre-load the datapoints into memory to avoid any data input bottleneck. The final RTSP value is the average over the total number of predictions. Note that further optimisations brought by some frameworks (e.g. NVIDIA's DeepStream) could be used to increase the speed of computation. However, as such frameworks do not cope well with compressed inputs, we did not use them for fairness of comparison.

### C. Evaluation on the synthetic dataset

We aim to evaluate the generalization capabilities of the proposed architectures w.r.t the flow orientation and number of flows. We consider orientation angles in [0°-315°] and the number of flows in $\{1, 2\}$).

*1) Accuracy and speed evaluation:* Obtained results are reported in Table III. They show high performance level for all models. The MAEs range from 0.17 to 0.21 with a correlation $R$ of 0.99. The best architecture being *DM CLS* at 0.17 of MAE. The three architectures have stable accuracies regardless of the number of flows and the orientation. This shows that the networks can be used seamlessly in different settings.

Regarding speed, we see that *DM CLF* is the slowest network with 811 streams processed in realtime. Such result was to be expected due to the sequential nature of the architecture, preventing to fully leverage the parallelization capabilities of the GPU. The fastest model is DM 3D that only uses 3D convolutions and exploits fully the parallelization capabilities of the GPU.

Overall, the results show small error with impressive speed, demonstrating the efficiency of the proposed approach.

TABLE III
ACCURACY AND SPEED RESULTS OF THE VARIOUS PROPOSED ARCHITECTURES ON THE MOVING DIGITS DATASET. THE "ORIENTATION" AND "FLOW NUMBER" COLUMS RESPECTIVELY DETAIL THE ACCURACY WHEN ONLY VARYING THE ORIENTATION AND THE NUMBER OF FLOWS. THE "AVG RESULTS" COLUMN SHOWS THE OVERALL RESULTS. RTSP STANDS FOR REALTIME STREAM PROCESSING (I.E. THE NUMBER OF STREAMS THAT CAN BE PROCESSED IN PARALLEL IN REALTIME).

| | Metric | Avg Results | Orientation | Flow number | RTSP |
|---|---|---|---|---|---|
| DM CLF | MAE | $0.20 \pm 0.02$ | $0.20 \pm 0.02$ | $0.21 \pm 0.03$ | 811 |
| | R | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ | |
| DM CLS | MAE | $\mathbf{0.17} \pm 0.01$ | $\mathbf{0.17} \pm 0.01$ | $\mathbf{0.14} \pm 0.01$ | 11,637 |
| | R | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ | |
| DM 3D | MAE | $0.19 \pm 0.02$ | $0.19 \pm 0.02$ | $0.17 \pm 0.01$ | **32,382** |
| | $R$ | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ | $0.99 \pm 0.00$ | |

*2) Prediction towards unseen configurations:* We now aim to study the generalisation capacities of the networks when the orientation or the number of flows varies.

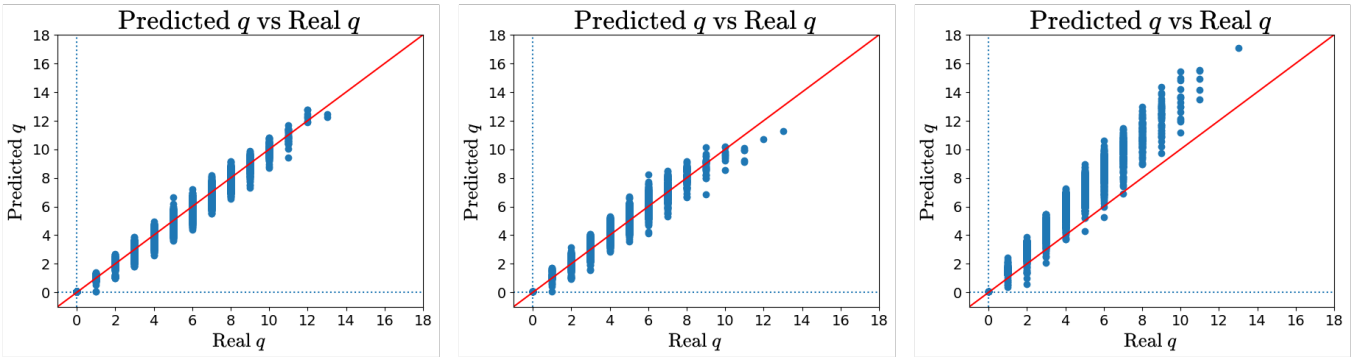
First, we train each network in a leave one out settings for each orientation (i.e. training on orientation angles from $0°$ to $270°$ and testing on $315°$, training on $45°$ to $315°$ and testing on $0°$, etc. ) and report the results in Table IV. The results show that the *DM CLF* and *DM CLS* architectures give the best performances with a MAE of 0.53 and 0.50 in average respectively. The least performing architecture is *DM 3D* with an averaged MAE of 0.85 and a standard deviation of 0.44. The three architectures well capture the trend of flow rate as the correlation coefficients are high.

Figure 8 further illustrates the obtained predictions. The first graph shows typical predictions when tested on a configuration similar to the training set. The second graph shows the typical results for the *DeepMotion CLF* and *DeepMotion CLS* in the leave-one-out setting. Finally, the last graph shows the worst prediction of the *DeepMotion 3D* architecture in the leave-one-out setting. In the latter case, the network overestimates the actual flow rate values, hence the slightly higher MAE.

Results relative to varying number of flows at training and testing stage are reported in Table V. Opposite to varying orientations, the trained networks fail to properly generalize. Overall, the MAE ranges from 1.41 to 3.21, which strongly differs from the results from Table IV. Such behavior was to be

TABLE IV
PREDICTION PERFORMANCES WHEN TRAINING ON ALL ORIENTATIONS BUT ONE ON MOVING DIGITS DATASET. EACH COLUMN IS THE ORIENTATION LEFT OUT AND TESTED UPON. BEST RESULTS ACCORDING TO EACH ORIENTATION ANGLE ARE MARKED IN BOLD FONT.

| | Metric | Average | All but one | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0° | 45° | 90° | 135° | 180° | 225° | 270° | 315° |
| DM CLF | MAE | $0.53 \pm 0.19$ | $\mathbf{0.42} \pm 0.06$ | $0.51 \pm 0.13$ | $0.52 \pm 0.07$ | $0.52 \pm 0.20$ | $0.90 \pm 0.21$ | $0.45 \pm 0.07$ | $0.43 \pm 0.09$ | $\mathbf{0.45} \pm 0.05$ |
| | R | $0.96 \pm 0.01$ | $0.96 \pm 0.01$ | $0.96 \pm 0.01$ | $0.95 \pm 0.01$ | $0.96 \pm 0.01$ | $0.96 \pm 0.01$ | $0.97 \pm 0.01$ | $0.97 \pm 0.01$ | $0.96 \pm 0.00$ |
| DM CLS | MAE | $\mathbf{0.50} \pm 0.16$ | $0.59 \pm 0.08$ | $\mathbf{0.37} \pm 0.05$ | $0.51 \pm 0.03$ | $\mathbf{0.40} \pm 0.02$ | $\mathbf{0.81} \pm 0.17$ | $\mathbf{0.37} \pm 0.04$ | $0.53 \pm 0.09$ | $0.45 \pm 0.06$ |
| | R | $0.97 \pm 0.00$ | $0.96 \pm 0.00$ | $0.97 \pm 0.00$ | $0.96 \pm 0.00$ | $0.97 \pm 0.00$ | $0.96 \pm 0.00$ | $0.97 \pm 0.00$ | $0.97 \pm 0.00$ | $0.97 \pm 0.00$ |
| DM 3D | MAE | $0.85 \pm 0.44$ | $0.93 \pm 0.05$ | $1.15 \pm 0.14$ | $\mathbf{0.36} \pm 0.07$ | $0.58 \pm 0.07$ | $1.61 \pm 0.30$ | $0.73 \pm 0.13$ | $\mathbf{0.30} \pm 0.03$ | $1.11 \pm 0.11$ |
| | R | $0.97 \pm 0.00$ | $0.98 \pm 0.00$ | $0.97 \pm 0.00$ | $0.97 \pm 0.00$ | $0.93 \pm 0.01$ | $0.97 \pm 0.00$ | $0.97 \pm 0.00$ | $0.98 \pm 0.00$ | $0.97 \pm 0.00$ |




(a) (b) (c)

Fig. 8. Prediction plots when training and testing on similar orientation (a) and in a leave-one-out setting (b) and (c). (a) a typical result when training and testing on the same orientation angle. (b) typical predictions by *DeepMotion CLF* and *DeepMotion CLS* architecture when trained in a leave-one-orientation-out manner. And, (c) is the worst result for the *DeepMotion 3D* in the leave-one-out setting on the orientations. Experiment was implemented on Moving Digits.

expected. As the networks are not forced to learn to detect the objects, adding or removing object flows is likely to change too drastically the inputs' distribution.

Overall these results empirically demonstrate that the proposed networks properly generalize towards unseen flow orientations. However, the current formulation shows some limitations in the case of varying number of flows.

TABLE V
PREDICTION ACCURACY WHEN TRAINING AND TESTING ON GENERATED DATASETS WITH VARYING NUMBER OF FLOWS (MOVING DIGITS). WHEN TESTING IS DONE ON 1 FLOW, TRAINING WAS DONE ON 2 (AND CONVERSELY). THE BEST ACCURACY FOR EACH COLUMN IS IN BOLD.

| | Metric | Average | Number of flows (test | |
|---|---|---|---|---|
| | | | 1 flow | 2 flows |
| DM CLF | MAE | $2.26 \pm 0.29$ | $1.86 \pm 0.29$ | $2.66 \pm 0.43$ |
| | R | $0.36 \pm 0.03$ | $0.44 \pm 0.12$ | $0.27 \pm 0.20$ |
| DM CLS | MAE | $3.21 \pm 0.21$ | $3.48 \pm 0.26$ | $2.93 \pm 0.45$ |
| | R | $0.68 \pm 0.04$ | $0.60 \pm 0.19$ | $0.75 \pm 0.16$ |
| DM 3D | MAE | $\mathbf{1.41} \pm 0.07$ | $\mathbf{1.26} \pm 0.22$ | $\mathbf{1.55} \pm 0.24$ |
| | R | $0.96 \pm 0.00$ | $0.97 \pm 0.00$ | $0.95 \pm 0.01$ |

### D. Evaluation on tunnel roads dataset

Hereafter, achieved performances by the proposed deep networks on UTCam dataset are reported and compared to the baseline RGB-based model.

*1) Accuracy and speed evaluation:* In the first experiment, compared models are trained and evaluated on samples issued from all cameras. Table VI summarizes the results.

The *DM CLS* architecture yields the best performances with 0.69 of MAE when compared with the RGB baseline (0.80 of MAE). The two other architectures, *DM CLF* and *DM 3D*, almost match up with the baseline.

Nevertheless, the proposed networks are much more efficient in term of memory and speed of processing. In particular, where the RGB baseline saturates all the VRAM of the GPU, the *DeepMotion CLS* and *DeepMotion 3D* architectures take up $10\times$ less space. Moreover, while the RGB architecture can barely process three cameras in parallel with a GPU, the *DeepMotion CLS* network can handle 5,699 streams in parallel and the *DeepMotion 3D* network, 10,604, which amounts to a speed up of $\times 2065$ and $\times 3699$ respectively.

Overall, the results show that learning deep architectures solely based on MVs leads to competitive (or even better) accuracy and a drastic reduction in computation and memory requirements.

TABLE VI
PREDICTION ACCURACY AND INFERENCE SPEED WHEN TRAINING ON ALL CAMERAS OF THE UTCAM DATASET. MIB STANDS FOR MEBIBYTE AND RTSP FOR REALTIME STREAM PROCESSING.

| Network | Metric | Results | # of weights | GPU (MiB) | RTSP |
|---|---|---|---|---|---|
| RGB baseline | MAE | 0.80 | 26.29 M | 7,805 | 2.76 |
| | R | 0.9 | | | |
| DM CLF | MAE | $0.81 \pm 0.03$ | 0.08 M | 2,753 | 667 |
| | R | $0.89 \pm 0.01$ | | | |
| DM CLS | MAE | $\mathbf{0.69} \pm 0.02$ | 0.46 M | **675** | 5,700 |
| | R | $0.93 \pm 0.00$ | | | |
| DM 3D | MAE | $0.83 \pm 0.02$ | 0.36 M | **675** | **10,210** |
| | R | $0.89 \pm 0.01$ | | | |

*2) Generalisation analysis:* We investigate the generalization abilities of the proposed networks. To do so, experiments

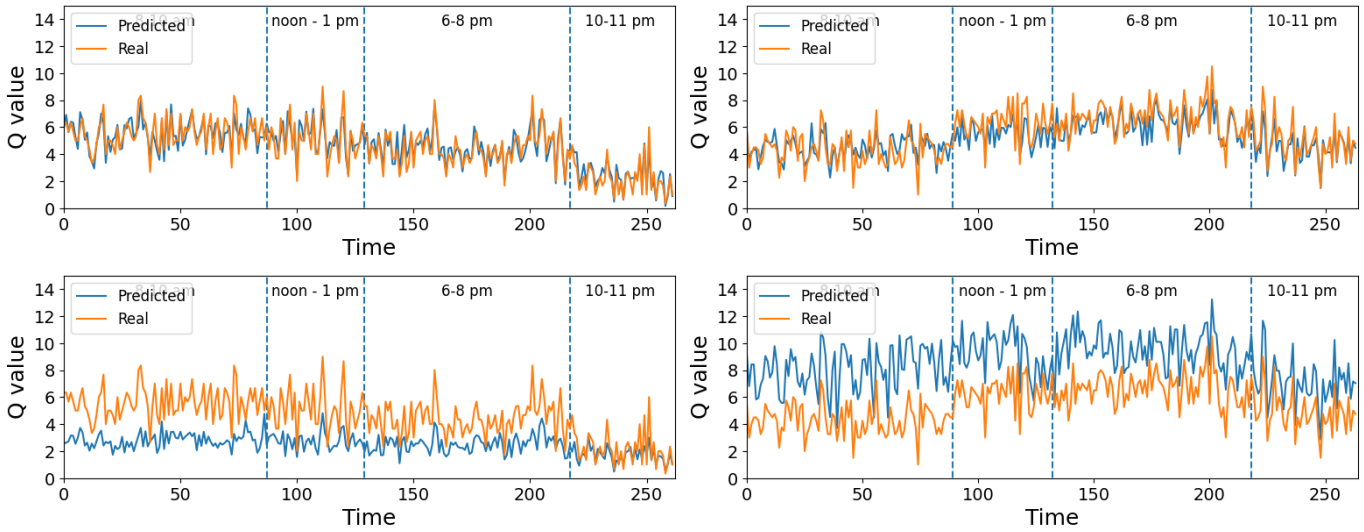



Fig. 9. Prediction plots over time for cameras $G2_4$ and $G3_3$ (first and second column) with the *DeepMotion 3D* network on UTCam. The top panels show the predictions for the two cameras when trained and tested on themselves (day 1 for training and day 2 for testing). The second line shows the results in the leave one out setting (similar camera orientation).

TABLE VII
MEAN ABSOLUTE ERRORS AND CORRELATION COEFFICIENTS (BETWEEN REAL AND PREDICTED VALUES) WHEN TRAINING ON ALL THE CAMERAS BUT ONE AND PREDICTING THE FLOW RATE ON THE CAMERA LEFT OUT. THE COLUMNS ARE THE ERRORS FOR THE LEFT OUT CAMERA. BEST RESULTS ARE IN BOLD.

| | Metric | All but one | | | | | All but one (Coming) | | | All but one (Going) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $G1_3$ | $G2_4$ | $G3_3$ | $C1_2$ | $C2_4$ | $G1_3$ | $G2_4$ | $G3_3$ | $C1_2$ | $C2_4$ |
| DM CLF | MAE | $\mathbf{1.69 \pm 0.49}$ | $1.50 \pm 0.27$ | $\mathbf{0.88 \pm 0.21}$ | $3.64 \pm 0.60$ | $2.65 \pm 0.55$ | $\mathbf{2.87 \pm 0.36}$ | $\mathbf{1.20 \pm 0.10}$ | $1.70 \pm 0.31$ | $\mathbf{2.60 \pm 0.47}$ | $4.46 \pm 0.93$ |
| | R | $0.78 \pm 0.03$ | $0.54 \pm 0.09$ | $0.76 \pm 0.11$ | $0.65 \pm 0.13$ | $0.54 \pm 0.11$ | $0.67 \pm 0.06$ | $0.42 \pm 0.09$ | $0.66 \pm 0.08$ | $0.64 \pm 0.13$ | $0.34 \pm 0.14$ |
| DM CLS | MAE | $2.61 \pm 0.33$ | $\mathbf{1.34 \pm 0.15}$ | $1.52 \pm 0.15$ | $3.84 \pm 0.38$ | $\mathbf{1.90 \pm 0.21}$ | $3.27 \pm 0.26$ | $1.43 \pm 0.07$ | $\mathbf{1.48 \pm 0.15}$ | $5.66 \pm 1.13$ | $\mathbf{3.20 \pm 0.38}$ |
| | R | $0.82 \pm 0.02$ | $0.35 \pm 0.13$ | $0.85 \pm 0.01$ | $0.72 \pm 0.05$ | $0.53 \pm 0.04$ | $0.77 \pm 0.08$ | $0.08 \pm 0.11$ | $0.83 \pm 0.04$ | $0.48 \pm 0.35$ | $0.47 \pm 0.12$ |
| DM 3D | MAE | $2.48 \pm 0.07$ | $1.65 \pm 0.38$ | $1.45 \pm 0.16$ | $\mathbf{3.19 \pm 0.34}$ | $2.81 \pm 0.46$ | $3.60 \pm 0.21$ | $2.15 \pm 0.55$ | $1.56 \pm 0.28$ | $4.79 \pm 0.39$ | $3.96 \pm 0.47$ |
| | R | $0.75 \pm 0.03$ | $0.70 \pm 0.04$ | $0.82 \pm 0.01$ | $0.66 \pm 0.02$ | $0.59 \pm 0.04$ | $0.71 \pm 0.03$ | $0.78 \pm 0.02$ | $0.82 \pm 0.04$ | $0.43 \pm 0.07$ | $0.40 \pm 0.08$ |

in the leave one camera out setting are performed. Two types of experiments are implemented: (i) either all cameras, regardless of their orientation, are considered, or (ii), the cameras are grouped by orientation (coming or going). The results under these two settings are in Table VII.

Regarding setting when all the cameras are considered regardless of the orientation, in a leave-one-out fashion, the results show that the proposed models do not properly generalize to the unseen camera. However, we can notice that the cameras $G2_4$ and $G3_3$ seem to provide with the best results overall. This can be explained by the fact that these two cameras are operating in almost similar conditions (c.f Figure 7) and therefore benefit from one another at prediction stage. Notice that $G2_4$ and $G3_3$ differ in number of lanes (respectively 4 and 3), which could hinder the generalization power of the models. Indeed, prediction in case of varying number of flows has proven a difficult task in the Moving Digits experiment.

We discuss the setting when the cameras are grouped by orientation. The expectation of such experiment is to improve the accuracy of the networks trained on data with alike distributions (avoid differences in noise such as headlights etc.). As before, we observe limited prediction accuracies, with better results on the cameras $G2_4$ and $G3_3$. Figure 9 further details the results on camera $G2_4$ and $G3_3$. The first line shows the prediction over time for the two cameras when trained and tested on themselves (day 1 for training and day 2 for testing). In such setting, we observe extremely good predictions. The second line shows the results in the leave-one-out setting (similar orientation, *DeepMotion 3D*). We can see that predictions for camera $G2_4$ are underestimated and that predictions for camera $G3_3$ are overestimated. However, we can also notice that the network manages to correctly follow the targeted flow rate trend, with similar peaks and pits at prediction time. Given that the cameras are alike in orientation, such results hint towards a problem linked with the changing number of lanes. As a lane is removed or added, the network wrongly estimates the final flow rate value.

Finally, comparing the two experimental settings, we can notice that training with more cameras, even if they have dissimilar orientations, helps improve the overall accuracy. Such results indicate that the proposed networks should be able to properly generalize to unseen cameras if more cameras were to be added to the dataset.

*3) Synthesis:* Looking at the results from subsubsection V-D1 *DeepMotion CLS* architecture provides with the best speed *vs* accuracy trade-off. Indeed, it is the more accurate model out of the three, both in terms of MAE and R, while, at the same time, providing with an impressive speed up of $\times 2065$ when compared with the RGB baseline (see Table VI).

subsubsection V-D2 further confirms these results. Although all three models have limited generalisation capabilities towards unseen configurations, the *DeepMotion CLS* architecture

clearly has the edge over the two others with respectively 4 and 6 of the best MAE and R out of 10 testing configurations (see Table VII).

## VI. Conclusion

We presented an end-to-end deep learning approach for the estimation of traffic flow rate leveraging the flow information present in the MPEG4 part-2 compressed videos. In order to train the proposed networks, we also introduce two datasets: Moving Digits, composed of generated videos, and UTCam, based on recordings from five surveillance cameras from Paris' tunnels. Our experiments show impressive speed gains when compared with a classical RGB detection-tracking-estimation method, while improving the accuracy. Moreover, the non frame-based architectures (*DeepMotion CLS* and *DeepMotion 3D*) require little memory and hence, perfectly fit industrial constraints.

In future work, we plan to explore the use of part of the residual information to provide the networks with texture information and help them better generalize to unseen cameras. A more general direction is to extend the use of the compressed motion information to other vision related task. In particular we expect large speed up gain for object detection as the moving objects are extracted *per design* by the MPEG4 part-2 compression.

## References


[1] L. Immers, S. Logghe, Traffic flow theory, Faculty of Engineering, Department of Civil Engineering, Section Traffic and Infrastructure, Kasteelpark Arenberg 40 (21).

[2] Y. Cho, J. Rice, Estimating velocity fields on a freeway from low-resolution videos, IEEE Transactions on Intelligent Transportation Systems 7 (4) (2006) 463–469. doi:10.1109/TITS.2006.883934.

[3] T. N. Schoepflin, D. J. Dailey, Algorithms for calibrating roadside traffic cameras and estimating mean vehicle speed, in: 2007 IEEE Intelligent Transportation Systems Conference, 2007, pp. 277–283. doi:10.1109/ITSC.2007.4357806.

[4] M. Bernaś, Objects detection and tracking in highly congested traffic using compressed video sequences, in: L. Bolc, R. Tadeusiewicz, L. J. Chmielewski, K. Wojciechowski (Eds.), Computer Vision and Graphics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 296–303.

[5] M. Naphade, D. C. Anastasiu, A. Sharma, V. Jagrlamudi, H. Jeon, K. Liu, M.-C. Chang, S. Lyu, Z. Gao, The nvidia ai city challenge, in: Prof. SmartWorld, Santa Clara, CA, USA, 2017.

[6] M. Naphade, M.-C. Chang, A. Sharma, D. C. Anastasiu, V. Jagarlamudi, P. Chakraborty, T. Huang, S. Wang, M.-Y. Liu, R. Chellappa, J.-N. Hwang, S. Lyu, The 2018 nvidia ai city challenge, in: Proc. CVPR Workshops, 2018, pp. 53—60.

[7] M. Naphade, Z. Tang, M.-C. Chang, D. C. Anastasiu, A. Sharma, R. Chellappa, S. Wang, P. Chakraborty, T. Huang, J.-N. Hwang, S. Lyu, The 2019 ai city challenge, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2019, p. 452–460.

[8] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, L. Zheng, A. Sharma, R. Chellappa, P. Chakraborty, The 4th ai city challenge, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020, p. 2665–2674.

[9] M. Tran, T. Dinh-Duy, T. Truong, V. Ton-That, T. Do, Q. Luong, T. Nguyen, V. Nguyen, M. N. Do, Traffic flow analysis with multiple adaptive vehicle detectors and velocity estimation with landmark-based scanlines, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2018, pp. 100–1007. doi:10.1109/CVPRW.2018.00021.

[10] Z. Tang, G. Wang, H. Xiao, A. Zheng, J.-N. Hwang, Single-camera and inter-camera vehicle tracking and 3d speed estimation based on fusion of visual and semantic features, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2018.

[11] H. Shi, Z. Wang, Y. Zhang, X. Wang, T. Huang, Geometry-aware traffic flow analysis by detection and tracking, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2018, pp. 116–1164. doi:10.1109/CVPRW.2018.00023.

[12] X. D. Yu, Ling-Yu Duan, Qi Tian, Highway traffic information extraction from skycam mpeg video, in: Proceedings. The IEEE 5th International Conference on Intelligent Transportation Systems, 2002, pp. 37–42. doi:10.1109/ITSC.2002.1041185.

[13] X. Yu, P. Xue, L. yu Duan, Q. Tian, An algorithm to estimate mean vehicle speed from mpeg skycam video, Multimedia Tools and Applications 34 (2006) 85–105.

[14] F. Li, F. Porikli, X. Li, Traffic congestion estimation using hmm models without vehicle tracking, in: In IEEE Intelligent Vehicle Symposium, 2004, pp. 188–193.

[15] Information technology – Coding of audio-visual objects – Part 2: Visual, Standard, International Organization for Standardization, Geneva, CH.

[16] P. Wang, J. Lai, Z. Huang, Q. Tan, T. Lin, Estimating traffic flow in large road networks based on multi-source traffic data, IEEE Transactions on Intelligent Transportation Systems 22 (9) (2021) 5672–5683. doi:10.1109/TITS.2020.2988801.

[17] C. Zheng, X. Fan, C. Wen, L. Chen, C. Wang, J. Li, Deepstd: Mining spatio-temporal disturbances of multiple context factors for citywide traffic flow prediction, IEEE Transactions on Intelligent Transportation Systems 21 (9) (2020) 3744–3755. doi:10.1109/TITS.2019.2932785.

[18] A. Abadi, T. Rajabioun, P. A. Ioannou, Traffic flow prediction for road transportation networks with limited traffic data, IEEE Transactions on Intelligent Transportation Systems 16 (2) (2015) 653–662. doi:10.1109/TITS.2014.2337238.

[19] C. Chen, Z. Liu, S. Wan, J. Luan, Q. Pei, Traffic flow prediction based on deep learning in internet of vehicles, IEEE Transactions on Intelligent Transportation Systems 22 (6) (2021) 3776–3789. doi:10.1109/TITS.2020.3025856.

[20] R. Ke, Z. Li, S. Kim, J. Ash, Z. Cui, Y. Wang, Real-time bidirectional traffic flow parameter estimation from aerial videos, IEEE Transactions on Intelligent Transportation Systems 18 (4) (2017) 890–901. doi:10.1109/TITS.2016.2595526.

[21] R. Ke, Z. Li, J. Tang, Z. Pan, Y. Wang, Real-time traffic flow parameter estimation from uav video based on ensemble classifier and optical flow, IEEE Transactions on Intelligent Transportation Systems 20 (1) (2019) 54–64. doi:10.1109/TITS.2018.2797697.

[22] J. Shi, C. Tomasi, Good features to track, IEEE Conference on Computer Vision and Pattern Recognition (1994) 593–600.

[23] B. D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'81, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1981, p. 674–679.

[24] C. Tomasi, T. Kanade, Detection and tracking of point features, Tech. rep., International Journal of Computer Vision (1991).

[25] G.-G. Lee, B.-s. Kim, W.-Y. Kim, Automatic estimation of pedestrian flow, in: 2007 First ACM/IEEE International Conference on Distributed Smart Cameras, 2007, pp. 291–296. doi:10.1109/ICDSC.2007.4357536.

[26] A. Kumar, P. Khorramshahi, W.-A. Lin, P. Dhar, J.-C. Chen, R. Chellappa, A semi-automatic 2d solution for vehicle speed estimation from monocular videos, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2018.

[27] Z. Liu, W. Zhang, X. Gao, H. Meng, X. Tan, X. Zhu, Z. Xue, X. Ye, H. Zhang, S. Wen, E. Ding, Robust movement-specific vehicle counting at crowded intersections, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020.

[28] P. Bergmann, T. Meinhardt, L. Leal-Taixé, Tracking without bells and whistles, in: The IEEE International Conference on Computer Vision (ICCV), 2019.

[29] L. Yu, Q. Feng, Y. Qian, W. Liu, A. G. Hauptmann, Zero-virus: Zero-shot vehicle route understanding system for intelligent transportation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020.

[30] M. Chang, C. Chiang, C. Tsai, Y. Chang, H. Chiang, Y. Wang, S. Chang, Y. Li, M. Tsai, H. Tseng, Ai city challenge 2020 – computer vision for smart transportation applications, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2020, pp. 2638–2647. doi:10.1109/CVPRW50498.2020.00318.

[31] N. Bui, H. Yi, J. Cho, A vehicle counts by class framework using distinguished regions tracking at multiple intersections, in: Proceedings of

the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020.

[32] I. Brkić, M. Miler, M. Ševrović, D. Medak, An analytical framework for accurate traffic flow parameter calculation from uav aerial videos, Remote Sensing 12 (22). doi:10.3390/rs12223844.
URL https://www.mdpi.com/2072-4292/12/22/3844

[33] R. Tusch, F. Pletzer, A. Krätschmer, L. Böszörmenyi, B. Rinner, T. Mariacher, M. Harrer, Efficient level of service classification for traffic monitoring in the compressed video domain, in: 2012 IEEE International Conference on Multimedia and Expo, 2012, pp. 967–972. doi:10.1109/ICME.2012.101.

[34] C. Kas, M. Brulin, H. Nicolas, C. Maillet, Compressed domain aided analysis of traffic surveillance videos, in: 2009 Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), 2009, pp. 1–8. doi:10.1109/ICDSC.2009.5289345.

[35] Y. Sun, Z. Liu, Z. Pan, Intersection traffic flow counting based on hybrid regression model, in: 2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP), 2019, pp. 1–4. doi:10.1109/ICSIDP47821.2019.9173285.

[36] Y. Zhou, Y. Lei, S. Yang, T. Shao, D. Tian, J. Shi, A traffic flow estimation method based on unsupervised change detection, Multimedia Systems (2021) 1–9.

[37] Y. H. Fu, H. Sahli, X. Fa Dong, J. Wang, A high efficient system for traffic mean speed estimation from mpeg video, Artificial Intelligence and Computational Intelligence, International Conference on 3 (2009) 444–448. doi:10.1109/AICI.2009.358.

[38] Z. Wang, X. Liu, J. Feng, J. Yang, H. Xi, Compressed-domain highway vehicle counting by spatial and temporal regression, IEEE Transactions on Circuits and Systems for Video Technology 29 (1) (2019) 263–274. doi:10.1109/TCSVT.2017.2761992.

[39] K. Mbonye, F. Ferrie, Attentive visual servoing in the mpeg compressed domain for un-calibrated motion parameter estimation of road traffic, in: 18th International Conference on Pattern Recognition (ICPR'06), Vol. 4, 2006, pp. 908–911. doi:10.1109/ICPR.2006.281.

[40] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.

[41] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, CoRR abs/1409.1259. arXiv:1409.1259.
URL http://arxiv.org/abs/1409.1259

[42] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, W.-c. WOO, Convolutional lstm network: A machine learning approach for precipitation nowcasting, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 28, Curran Associates, Inc., 2015.
URL https://proceedings.neurips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, A. C. Berg, Ssd: Single shot multibox detector., in: B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), ECCV (1), Vol. 9905 of Lecture Notes in Computer Science, Springer, 2016, pp. 21–37.
URL http://dblp.uni-trier.de/db/conf/eccv/eccv2016-1.html#LiuAESRFB16

[44] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft, Simple online and realtime tracking, in: 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 3464–3468. doi:10.1109/ICIP.2016.7533003.

[45] R. E. Kalman, A new approach to linear filtering and prediction problems, Journal of Basic Engineering 82 (1) (1960) 35. doi:10.1115/1.3662552.
URL http://dx.doi.org/10.1115/1.3662552

[46] H. W. Kuhn, B. Yaw, The hungarian method for the assignment problem, Naval Res. Logist. Quart (1955) 83–97.

[47] N. Wojke, A. Bewley, D. Paulus, Simple online and realtime tracking with a deep association metric, in: 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3645–3649. doi:10.1109/ICIP.2017.8296962.

[48] Y. LeCun, C. Cortes, MNIST handwritten digit database [cited 2016-01-14 14:24:11].
URL http://yann.lecun.com/exdb/mnist/

## APPENDIX

We discuss key points induced by the annotation process. The collected videos are annotated using induction loop records. Tunnel videos are recorded at about 25 FPS by onsite coders. The traffic flow is recorded from induction loops and stored by Automatic Data Recording (ADR) stations in log files at fixed intervals of 20 seconds. As both video and traffic data come from separate sources, we need to synchronize them so as to annotate the videos with the corresponding traffic flow labels. However two main problems arise: the non-synchronization of the time clocks and inconsistencies in the videos frame rate.

In particular, coders and ADR stations time clocks are not synchronized, furthermore, each coder has its own clock. Therefore, the offset between each pair of coder/ADR station needs to be computed. Obviously the computation of such offset can only be done by matching the recorded flow rates with the visual video information. Luckily, the loop detectors compute the flow parameters for each lane, simplifying this matching. We use the proposed RGB baseline for offset calibration and verify by hand the truthfulness of the obtained offsets.

The second issue is the inconsistency of the video frame rate. Although the theoretical frame rate is of 25 FPS, in practice, this is not the case due to exploitation constraints (packet loss, etc.). Hence, when associating video recordings with flow rate values, not only do we need to compute the offsets between each ADR station and the associated camera, but we also must visually check that shifts do not occur in the stream. Such task is done in a semi-automated way using a script, which prompts the user for confirmation at given intervals. This methodology possibly implies a slight shift in the data, and therefore we choose to associate videos of 21 seconds with the recorded traffic flow measurements so as to ensure that each video encompasses the whole measurement period of flow parameters. Consequently, annotations might be noisy as more vehicles than the ones accounted for in the measurements may be visible on screen.