PUNE RESEARCH DISCOVERY ISSN 2455-9202
AN INTERNATIONAL JOURNAL OF ADVANCED STUDIES VOL 4, ISSUE 2

MULTI-CAMERA VEHICLE TRACKING USING VIDEO ANALYTICS

J.KAVITHA[1],MRS K.BALA[2], DR.D.RAJINIGIRINATH[3],
[1] P.G Student, [2]Asst.Prof, [3]Professor,
Department of Computer Science and Engg,
Sri Muthukumaran Institute Of Technology, Chikkarayapuram, Chennai,
Tamil Nadu-600069, **(TN) INDIA**

***Abstract*—Recently video-enabled camera will be the most pervasive type of sensor. Such cameras will enable continuous surveillance through *heterogeneous camera networks* consisting of fixed camera systems as well as cameras on mobile devices. The challenge in these networks is to enable *efficient video analytics*: the ability to process videos cheaply and quickly to enable searching for specific events or sequences of events. In this paper, we discuss the design and implementation of Kestrel, a video analytics system that tracks the path of vehicles across a heterogeneous camera network. In Kestrel, fixed camera feeds are processed on the cloud, and mobile devices are invoked *only* to resolve ambiguities in vehicle tracks. Kestrel's mobile device pipeline detects objects using a deep neural network, extracts attributes using cheap visual features, and resolves path ambiguities by careful association of vehicle visual descriptors, while using several optimizations to conserve energy and reduce latency. Our evaluations show that Kestrel can achieve precision and recall comparable to a fixed camera network of the same size and topology.**

***Index Terms*—Cyber-physical Systems, Video Analytics, Vehicle Trajectory Inference, Heterogeneous Camera Network**

I. INTRODUCTION

Video cameras will soon be, if they are not already, the most pervasive type of sensor in the Internet of Things. They are ubiquitous in public places, available on mobile devices and drones, in cars as dashcams, and on security personnel as bodycams. Such cameras are valuable because they provide rich contextual information, but pose a challenge for the very same reason: it requires significant manual effort to extract meaningful semantic information from these videos. To address this, recent research [20] has started exploring the design of *video analytics* systems, which automatically process videos in order to extract semantic information. h*eterogeneous Camera Networks:* Future video analytics systems will need to operate on *heterogeneous camera networks*. These networks consist of fixed camera surveillance systems which can be engineered such that camera feeds can be

transmitted to a cloud-based processing system [20] also more constrained camera devices (mobile cameras, dash cams and body cams) that may be wirelessly connected, have limited rocessing power, and, in some cases, may be battery powered. The advantage of heterogeneous camera networks is that they can greatly increase accuracy and coverage. This is, by now, well established even in the public sphere. After the success in using mobile phone images and videos in the Boston marathon bombing a few years ago In particular, cities like Chicago [11] also have dash cams for recording traffic stops.

Video Analytics for Vehicle Tracking: As a first step to understanding how to design video analytics systems in heterogeneous camera networks, we take a specific problem: *to automatically detect a path taken by a vehicle through a heterogeneous network of non-overlapping cameras*. In our problem setting, each mobile or fixed camera either continuously or intermittently captures video, together with associated metadata (camera location, orientation, resolution, *etc*.). Conceptually, this creates a large corpus of videos over which users may pose several kinds of queries either in *near real-time*, or *after the fact*, the most general of which is: What path did a given car, seen at a given camera, take through the camera network? Commercial surveillance systems do not support automated multi-camera vehicle tracking, nor heterogeneous camera network. They either require a centralized collection of videos [3], or perform object detection on an individual camera, leaving it to a human operator to perform association of vehicles across cameras by manual inspection [1].

Contributions: This paper describes the design of Kestrel1 (§II), a video analytics system for vehicle tracking. Users of Kestrel provide an image of a vehicle (captured, for example, by the user inspecting video from a static camera), and the system returns the sequence of cameras (*i.e.*, the path through the camera network) at which this vehicle was seen. This system *carefully selects, and appropriately adapts, the right combination of vision algorithms to enable accurate end-toend tracking, even when individual components can haveless than perfect accuracy, while respecting mobile device constraints*. Kestrel addresses the challenge described above using one key architectural idea (Figure 1). Its *cloud pipeline* continuously processes videos streamed from fixed cameras to extract vehicles and their attributes, and when a query is posed,computes vehicle trajectories across these fixed cameras. *Only when there is ambiguity in these trajectories*, Kestrel queries one or more mobile devices (which runs a *mobile pipeline* optimized for video processing on mobile devices). Thus, mobile devices are invoked only when absolutely necessary, significantly reducing resource consumption on mobile devices.

Kestrel uses novel techniques for fast execution of deep (those with 20 or more layers) Convolutional Neural Networks(CNNs) on mobile device embedded GPUs (§II-A). These deep CNNs are more accurate, but mobile GPUs cannot execute these deep CNNs because of insufficient memory. By quantifying the memory sumption of each layer, we have designed a novel approach that offloads the bottleneck layers to the mobile device's CPU and pipelines frame processing without impacting the accuracy. Kestrel leverages these optimizations as an essential building block to run a deep CNN ([21]) on mobile GPUs for *detecting* objects and drawing bounding boxes around them on a frame.

Kestrel employs an accurate and efficient object *tracker* for objects within a single video (§II-B), enabling an order of magnitude more efficiency than per-frame object detection.

This tracker runs on the mobile device, and we use it to extract object attributes, like the direction of motion as well as object features for matching. Existing general purpose trackers are computationally expensive, or can only track single objects.Kestrel only periodically detects objects (thereby reducing the energy cost of object detection), say every *k* video frames, so that the tracking algorithm has to only be accurate in between frames at which detection is applied.

II. KESTREL DESIGN

Figure 1 shows the system architecture of Kestrel, which consists of a *mobile device pipeline* and a *cloud pipeline*.

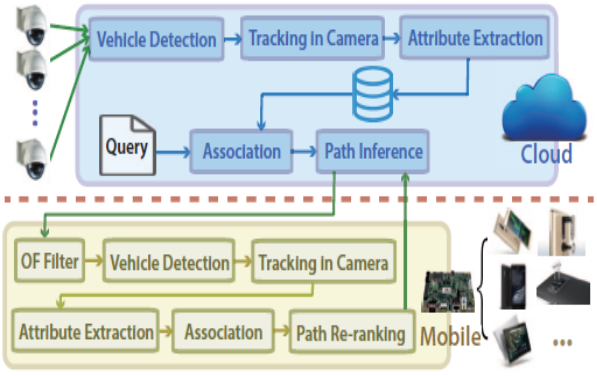

Fig. 1—Kestrel System Architecture

Videos from fixed cameras are streamed and stored on a cloud. Mobile devices *store videos locally* and *only send metadata to the cloud* specifying when and where the video is taken. Given a vehicle to track through the camera network, the cloud detects cars in each frame (*object detection*), determines the direction of motion of each car (*tracking*) in each single camera view and extracts visual descriptors for each car (*attribute extraction*). Then, across nearby cameras, it tries to *match* cars (*crosscamera association*), and uses these to infer the path the car took (*path inference*) *only* on videos from the fixed cameras.Only when the cloud determines that it has low confidence in the result, it *uses metadata from the mobile devices* to query relevant mobile devices *to increase tracking accuracy*. Once the mobile device receives the query, it performs roughly the same steps, but only in a small segment of the captured video (and some of these steps are optimized to save energy). More specifically, the cloud sends to the mobile device its inferred candidate paths, and the mobile pipeline *re-ranks* these paths. The outcome is increased confidence in the estimated path. In the following sections, we describe Kestrel's components: some components run on the cloud alone, others run on both the cloud and the mobile device.

A. Object Detection: Deep CNNs

Kestrel uses GPUs on mobile devices to run Convolutional Neural Networks (CNNs) for vehicle detection. Many mobile devices incorporate GPUs (like nVidia's TK1 and TX1 or Qualcomm's Adreno), including mobile phones such as Lenovo Phab 2 Pro [25] and Asus ZenFone AR [2], as well as tablets like the Pixel C [16], the Google's Tango [15]. Unfortunately, the memory requirements of deep CNNs are outpacing the growth in mobile GPU memory. The trend today is towards deeper CNNs (*e.g.*, ResNet has 152 layers), with increasing memory footprint. Kestrel carefully optimizes CNN memory footprint to enable state-of-the-art object detection on mobile GPUs.

The CPU offload Optimization: Prior work has considered offloading computation to overcome processor limitations [8, 18]. We explore offloading only the FC layer computation to the CPU, because, since the CPU supports virtual memory it can more effectively manage memory over-subscription required for the FC layer. With CPU offloading, we observe that when the CPU is running the FC layer on the first video frame,the GPU cores are idle. So we adopt a *pipelining* strategy to start running the second video frame on the GPU cores rather than letting them idle. To achieve this kind of pipelining, we run the FC layers on the CPU on a separate thread, as GPU computation is managed by the main thread.

Other memory optimizations: In our evaluation (§III-E), we compare offloading with pipelining with other memory optimizations that have been proposed in the literature. We explore *reducing the size* of the FC layer ([12] has explored similar techniques in different settings), which can potentially reduce detection accuracy. To reduce the size of the FC layer, obtain the output vector. Splitting this matrix multiplication and reading the weight matrix in parts allows us to reuse

memory.However, re-using memory and overwriting the used chunks incurs additional file access overheads.

B. Attribute Extraction

After detecting objects, Kestrel extracts two *attributes* of the object from a video, including its (i) direction of motion, and (ii) a low-complexity visual descriptor. These attributes are used to associate objects across cameras (§II-C). To estimate the direction of motion, it is important to *track* the object across successive video frames.

Light-weight Multi-Object Tracking: To be able to track objects2 in a video captured at a single camera, Kestrel needs to take objects detected in successive frames and associate them across multiple frames of the video. Our tracking algorithm has two functions: (i) it can reduce the number of times YOLO is invoked, which in turn reduces the latency and conserves energy, and (ii) it can be used to estimate direction of motion. The state-of-the-art object tracking techniques [13] take as input the pre-computed bounding box of the object, then extract sophisticated features like SIFT [9], SURF [17], *etc.*from the bounding box, and then iteratively infer the position of these key-points in subsequent frames using feature matching algorithms like RANSAC .

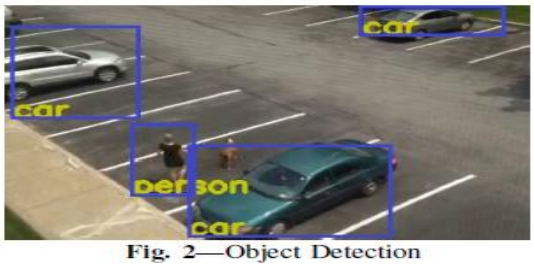

Fig. 2—Object Detection

Object State Extraction: An object seen in a single camera can go through different

states: it can *enter* the scene,*leave* the scene, *move* or be *stationary*. Determining these states is important to filter out uninteresting objects like permanently parked cars, etc. Kestrel differentiates moving objects from stationary

ones by detecting the difference of the optical flow in the bounding boxes and in the background. For moving objects, the state of the object changes from entering, moving, to exit, as it moves through the camera scene. A moving object can become stationary in the scene, a stationary object can also start moving at any time. For example, a car may

come to a stop sign or a traffic light at an intersection, or park temporarily for loading passengers. If the object is detected as *exited* from the scene, then the object can be evicted. Kestrel uses *fast eviction*: after an object exits the scene, all the feature points and information for tracking are evicted from memory. This, not only helps save memory, but also improves accuracy because evicting the keypoints from previous objects can reduce the search space for matching candidates (§II-C). *Extracting the Direction of Motion:* Kestrel estimates the direction of motion of a vehicle to on a mobile device to eliminate ambiguity. Suppose a vehicle is moving towards camera A as shown in Figure 5, but one of the cloud-supplied paths shows that it went through a different camera, say B located away from the vehicle's direction of motion. Since it could not possibly have gone through B, the mobile pipeline can lower that path's rank (§II-D). Extracting the direction poses two challenges: how to deal with the *movement of the mobile device*, and how to *transform the direction of motion* to a global coordinate frame.

Kestrel addresses the first challenge by *compensating for camera movement*. In general, to extract the direction of motion in the frame coordinates, we can use the difference between the bounding box positions from consecutive frames. In our experience, however, using the optical flow in the bounding box gives us more fine-grained direction estimation that is robust to the errors in YOLO's bounding boxes.

Fig. 3—Motion Analysis via Multi-Object Tracking

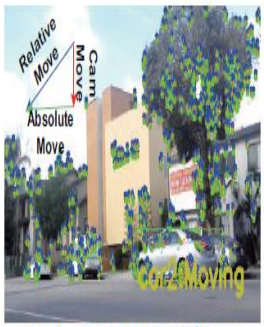

Fig. 4—Camera Movement Subtraction

Fig. 5—Extracting the Direction of Motion and Coordinate Transformation

To compensate for camera movement, we subtract the optical flow of the background from the optical flow of the object bounding box. The result is the direction of the object (Figure 4).Kestrel needs to transform the direction of motion to a global coordinate frame in order to reason about, and re-rank, other vehicle paths provided by the cloud. For an arbitrary camera orientation, the exact moving direction in global coordinates can be calculated using a homography transformation [24].However, in most practical scenarios, cameras have a small pitch and roll (*i.e.*, no tilt towards ground or sky and no rotation, videos are often recorded in either portrait or landscape)

especially across a small number of frames. So, we only use the azimuth of the camera, and infer only the horizontal direction of moving objects (Figure 5). As per the Android convention (that we use to obtain our dataset), right (east) is 0◦ and counterclockwise is positive in camera (global) coordinates. The transformation is simply $\theta g = \gamma + (\theta c - 90^\circ)$ where θc (θg) is the

direction of motion in the camera (global) coordinates, and γ is the azimuth of the camera orientation.To obtain the azimuth for mobile cameras we use the orientation sensor on off-the-shelf mobile devices. We have a custom Android application that records meta-data of the video (GPS, orientation sensor, accelerometer data, *etc*.) along with the video. However, to use this information directly .

Extracting the Direction of Motion: Kestrel estimates the direction of motion of a vehicle to on a mobile device to eliminate ambiguity. Suppose a vehicle is moving towards camera A as shown in Figure 5, but one of the cloud-supplied paths shows that it went through a different camera, say B located away from the vehicle's direction of motion. Since it could not possibly have gone through B, the mobile pipeline can lower that path's rank (§II-D). Extracting the direction poses two challenges: how to deal with the *movement of the mobile device*, and how to *transform the direction of motion* to a global coordinate frame.

Kestrel addresses the first challenge by *compensating for camera movement*. In general, to extract the direction of motion in the frame coordinates, we can use the difference between the bounding box positions from consecutive frames. In our experience, however, using the optical flow in the bounding box gives us more fine-grained direction estimation that is robust to the errors in YOLO's bounding boxes. To compensate for camera movement, we subtract the optical flow of the background from the optical flow of the object bounding box.

The result is the direction of the object (Figure 4). Kestrel needs to transform the direction of motion to a global coordinate frame in order to reason about, and re-rank, other vehicle paths provided by the cloud. For an arbitrary camera orientation, the exact moving direction in global coordinates can be calculated using a homography transformation .However, in most practical scenarios, cameras have a small pitch and roll (*i.e.*, no tilt towards ground or sky and no rotation, videos are often recorded in either portrait or landscape) especially across a small number of frames. So, we only use the azimuth of the camera, and infer only the horizontal direction of moving objects (Figure 5).

.

IV. RELATED WORK

DNNs on mobile devices: Recent work has explored neural nets on mobile devices for audio sensing activity detection, emotion recognition and speaker identification [21]. Their networks use only a small number of layers and are much simpler than the networks required for image recognition tasks. DeepX [23] is able to achieve reduced memory footprint of the deep models via using compression, at the cost of a small loss in accuracy. LEO [26] schedules multiple sensing applications on mobile platforms efficiently. MCDNN [11] explores cloud offloading and on-device versus cloud execution tradeoff, but on models smaller than the ones required for our work. Finally, DeepMon [16] proposes offloading the FC layers to the GPU for low power, lower frame rate (1-2 fps) applications, while using a tensor decomposition technique to reduce the memory footprint, at the expense of accuracy.

Object Detection: Early object detectors use deformable part models [15] or cascade classifiers [17], but perform relatively poorly compared to recent CNN based classificationonly schemes which achieve high

accuracy at real-time speeds. However, top detection systems like Fast R-CNN [18] exhibit less than real-time performance even on server-class machines. YOLO is a one-shot CNN-based detection algorithm that

predicts bounding boxes and classifies objects, and we have used a mobile GPU on a deep CNN like YOLO.

Object Tracking: Prior tracking approaches like blob tracking [19] work well in static camera networks, but not for mobile cameras. Many other trackers are targeted to single object tracking,

V. CONCLUSION

This paper explores whether it is possible to perform complex visual detection and tracking by leverprovements in mobile device capabilities. Our system, Kestrel,tracks vehicles across a heterogeneous multi-camera network by carefully designing a combination of vision and sensor processing algorithms to detect, track, and associate vehicles across multiple cameras. Kestrel achieves > 90% precision and recall on vehicle path inference, while significantly reducing energy consumption on the mobile device.Future work includes experimenting with Kestrel at scale with different traffic densities and camera placement densities than the ones we have explored in this paper. We anticipate that our results will be insensitive to camera placement density (because our approach only detects paths through the camera network), but may perform differently at different traffic densities because of inaccuracies in the underlying computer vision tools. Other future work includes extending Kestrel to support more queries, and different types of objects, so it can be used as a general visual analytics platform.

REFERENCE

[1]. *Agent VI*. http://www.agentvi.com/.
[2]. *Asus ZenFone AR*. https://www.asus.com/us/Phone/ZenFone-AR-ZS571KL.
[3]. *Avigilon Video Analytics*. http://avigilon.com/products/video-analytics/solutions/.
[4]. A.Viterbi. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm". In:*IEEE Trans. Inf. Theor.* 13.2 (Sept. 2006).
[5]. B.Lucas and T.Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In:IJCAI'81.
[6]. C.Rother, V.Kolmogorov, and A.Blake. ""GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts". In: SIGGRAPH '04.
[7]. *Data Acquisition Kit*. http://www.dataq.com/products/di-149/
[8]. .D.Chu et al. "Balancing Energy, Latency and Accuracyfor Mobile Sensor Data Classification". In: *Proc. ofSensys 11*
[9]. .Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2(Nov. 2004).
[10]. E.Mark et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *Int. J. Comput. Vision* 88.2 (June 2010).
[11]. *Every Chicago patrol officer to wear a body camera by2018*.
[12]. G.Chen, C.Parada, and G.Heigold. "Small Footprint keyword spotting using deep neural networks". In: *Proc.*

of IEEE ICASSP '14.

[13]. G.Nebehay and R.Pflugfelder. "Clustering of Static-Adaptive Correspondences for Deformable Object racking".

In: *CVPR '15*. IEEE.

[14]. *Google Maps Directions API*.
[15]. *Google Tango*. https://www.google.com/atap/projecttango/.
[16]. *Google's Pixel C*. https://store.google.com/product/pixel_c.
[17]. H.Bay, T.Tuytelaars, and L.Van Gool. "SURF: SpeededUp Robust Features". In: ECCV'06.
[18]. H.Eom et al. "Machine Learning-Based Runtime Schedulerfor Mobile Offloading Framework". In: *Proc. ofUCC '13*.
[19]. H.Marko and P.Matti. "A texture-based method for modeling the background and detecting moving objects".In: *IEEE Trans. on PAMI* 28.4 (2006), pp. 657–662.
[20]. H.Zhang et al. "Live Video Analytics at Scale with Approximation and Delay-Tolerance". In: *NSDI '17*.
[21]. J.Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CVPR '16*.
[22]. J.Shi and C.Tomasi. "Good features to track". In: *CVPR*.IEEE, 1994, pp. 593–600.
[23]. K.Alex. "One weird trick for parallelizing convolutional neural networks". In: *CoRR*abs/1404.5997 (2014).
[24]. Leal-Taixé et al. "MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking". In:*arXiv:1504.01942 [cs]* (Apr. 2015). *Lenovo Phab 2 Pro*. http : / /www3. lenovo . com / us /en/smart- devices/ - lenovo-smartphones/phab- series /Lenovo-Phab-2-Pro.