Classification of crash and near-crash events from dashcam videos and telematics data

Leonardo Taccari, Francesco Sambo, Luca Bravi, Samuele Salti
Leonardo Sarti, Matteo Simoncini, Alessandro Lori
Verizon Connect
via Paisiello 16-20, 50144 Florence, Italy
Email: leonardo.taccari@verizonconnect.com

***Abstract*—The identification of dangerous events from sensor data is a fundamental sub-task in domains such as autonomous vehicles and intelligent transportation systems. In this work, we tackle the problem of classifying crash and near-crash events from dashcam videos and telematics data. We propose a method that uses a combination of state-of-the-art approaches in computer vision and machine learning. We use an object detector based on convolutional neural networks to extract semantic information about the road scene, and generate video and telematics features that are fed to a random forest classifier. Computational experiments on the SHRP2 dataset show that our approach reaches more than 0.87 of accuracy on the binary problem of distinguishing dangerous from safe events, and 0.85 on the 3-class problem of discriminating between crash, near-crash, and safe events.**

I. Introduction

In recent years, the diffusion of connected vehicles and the push for autonomous vehicles has led to an enormous increase in the kind and amount of data that can be collected from a vehicle. Vehicles can be remotely monitored with fleet management solutions for commercial purposes, or for safety or insurance reasons in consumer vehicles. In particular, nowadays most connected vehicles include not only the standard telematics sensors, such as accelerometers and gyroscopes, but also video cameras and other sensors such as radar or lidar. The availability of this kind of data spurs the investigation of new accurate method of analyzing and understanding the semantics of road scenes in video footage, and fusing this knowledge with data from other sensors.

Identifying dangerous situations in driving behaviors, with the aim to prevent their occurrence and provide a post-hoc review mechanism, is a problem at the very core of intelligent transportation systems research. Given vehicle data collected from on-board sensors, the problem of crash and near-crash event identification calls for the detection and classification of the severity of the driving events involving the ego vehicle, i.e., the vehicle where the sensors are mounted.

The identification of crash and near-crash events is actually a two-step problem: it involves the *detection* of significant events, and their *classification*, i.e., the association of a semantic label that represents the type or severity of the event. In this article, we focus specifically on the problem of classifying driving events from videos collected by a front-facing dashboard camera (dashcam) and telematics data.

In the literature, road activity recognition and accident detection from videos has been mainly studied in still images or videos from surveillance cameras. The work in [1] proposes an algorithm based on Markov random fields to recognize behavior patterns for vehicles appearing in traffic images at intersection. A similar context is considered in [2], where the authors present a vision-based crash detection algorithm and a system for recording and reporting traffic accidents at intersections. In [3], the problem of classifying still images at intersections into a set of atomic scenes is investigated. Of particular interest is the work in [4], where the authors consider the problem of anticipating accidents from dashcam videos. The method they use is based on recent deep learning techniques: specifically, they use an object detection algorithm followed by a recurrent spatio-temporal attention mechanism that points, for each time step, to the object(s) most likely to be involved in a crash. Although the videos are collected from on-board dashcams, this work is again focusing on events involving third-party subjects only. Considering other kinds of data, in [5] traffic accident detection is tackled from mobile phone sensors, such as accelerometers and acoustic data. Another work that aims at integrating data from multiple on-board sensors is the one in [6], where the authors present an analysis framework based on a deep learning solution to classify distracted driving behavior from an interior-facing camera and telematics data.

In this paper, we use a set of manually annotated events from the SHRP2 dataset [7] as the source of ground truth labels for classification of crash, near-crash or safe events occurring to the ego vehicle. Our procedure consists of a sequence of steps that lead to the extraction of a rich set of high-level features. Then, we use a Random Forest classifier (RF, [8]) to combine the features from the video domain and from the telematics data, and classify the event. Experimental results show that our system is able to classify effectively crash, near-crash and normal events.

The paper is organized as follows: Section II presents the dataset we used to build and validate our system, Section III describes in detail the different components of the system, Section IV shows experimental results and Section V draws conclusions and future research directions.

II. Dataset

To develop and validate our method, we use a dataset extracted from the Second Strategic Highway Research Program (SHRP2) Naturalistic Driving Study database [7], which originates from a research effort coordinated by the Virginia Tech Transportation Institute. The participants of the study are more than 2000 drivers recruited in six sites across the U.S. (Indiana, Pennsylvania, Florida, New York, North Carolina, Washington). The entire database contains more than 5 million trip files, with data from multiple on-board sensors, such as front and interior-facing cameras, accelerometer, and radar.

In this article, we concentrate on events that were automatically detected with a set of trigger algorithms based on telematics sensor data, followed by a manual validation by human annotators [7]. For each detected event, the SHRP2 dataset contains a video and a stream of telematics data for a 30 second segment, as well as the timestamp of start and end of the actual event. The event always starts at the 20th second, while the end typically occurs a few seconds later. The videos have a framerate of 15 frame per second and a 480×356 resolution, while the telematics data contains instant speed and 3-axis acceleration data sampled at 10Hz.

Of all the detected events in SHRP2, our subset contains 280 crash events, 6755 near-crash events, and 4082 baseline videos, that were labeled as safe events. According to the definitions in [7], crash events are defined as "events that involve any contact of the subject vehicle with an object, either moving or fixed, at any speed in which kinetic energy is measurably transferred or dissipated". On the other hand, near-crash events are defined as "any circumstance that requires a rapid evasive maneuver by the subject vehicle, or any other vehicle, pedestrian, cyclist, or animal, to avoid a crash". A rapid evasive maneuver can involve steering, braking, accelerating, or any combination of control inputs. Finally, the baseline, safe videos may contain "any incident or manoeuvre within the bounds of normal driving behaviors and scenarios that is accurately represented by the telematics data. The driver may react to situational conditions and events, but the reaction is not evasive and the situation does not place the subject or others at elevated risk".

It is worth stressing again that the safe events are not randomly selected: indeed, they were detected by the automatic SHRP2 trigger algorithms, and only later discarded by the human annotators based on visual inspection. They contain non-risky driving, but typically with mild to strong braking or acceleration.

III. Methods

Our procedure for harsh event classification is represented in Figure 1. For each candidate event, we apply a Convolutional Neural Network model [9] to detect relevant road objects in each frame of the video, and we compute a dense optical flow for each two consecutive frames. Then, we run a pipeline that computes a set of intermediate features, that are passed to the feature extractor. Additional groups of features are extracted also directly from the optical flow vectors and from the telematics data. Finally, the full set of features is used, on the training set, to train a Random Forest model according to the ground truth labels derived from the SHRP dataset. The trained RF can then be used to predict the class of an event on test data.

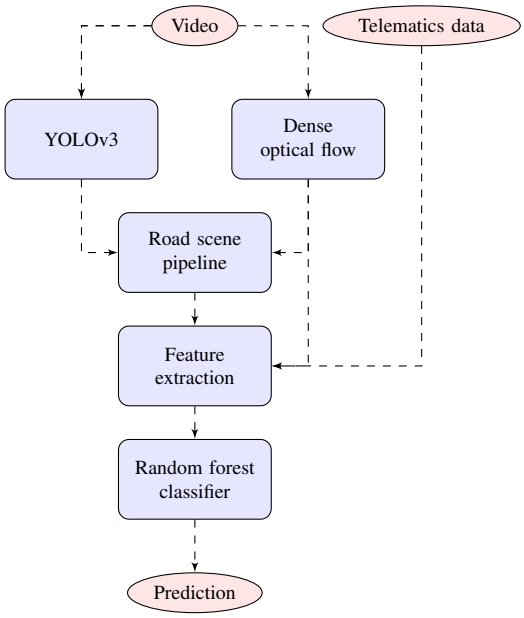

Fig. 1. Diagram of the classification procedure.

A. Object detection with convolutional neural networks

We perform object detection by means of a state-of-the-art convolutional neural network architecture. We use a modified version of YOLOv3 [10], trained on the COCO [11] dataset, but only used to predict bounding boxes for objects belonging to the following 8 classes:

- car
- truck
- bus
- motorbike
- bicycle
- person
- traffic light
- stop sign.

The input to the network is a square color image with the 3 RGB channels. The architecture of YOLOv3 consists of a sequence of 53 convolutional layers [12], with some shortcut connections as in residual networks [13], to let the gradient propagate more easily. Each convolutional block consists of:

1) A convolutional filter with kernel of size 3×3 or 1×1;
2) A non-linear activation function, namely a Leaky Rectified Linear Unit [14], which is defined as

$$\phi(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.1x & \text{if } x < 0; \end{cases}$$

3) A max-pooling layer, of size 2×2 and stride 2.

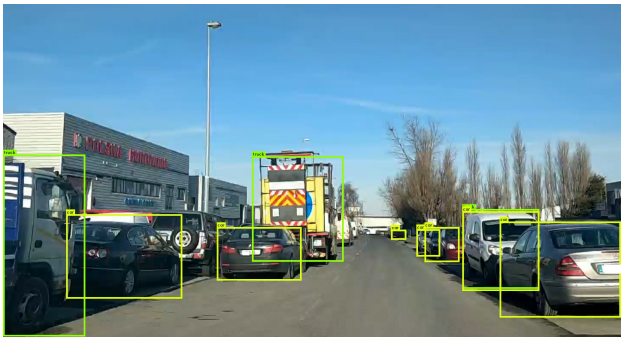

Fig. 2. Example of the output of YOLOv3 416×416 object detection on an image depicting a road scene. The detector is quite accurate, but a few mistakes can be noticed: for instance, a car on the left (partially occluded) is not detected.

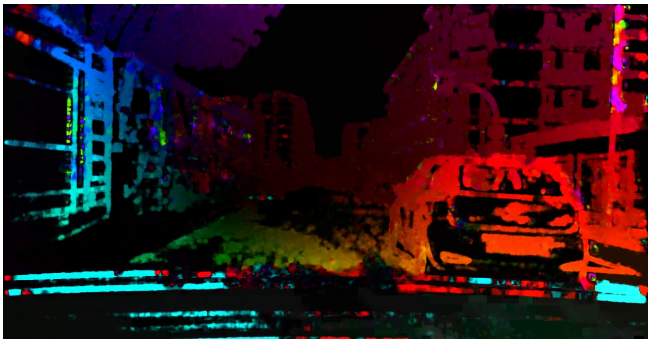

Fig. 3. Example of optical flow between two consecutive frame. The hue of a pixel denotes the direction of the flow in that point, while the intensity encodes its magnitude. A bus and a car can be seen on the left and on the right of the image, respectively.

The prediction is then carried out at 3 different scales thanks to a number of final upsampling convolutional layers, similarly to the idea of feature pyramid networks [15], that are instrumental in improving the classification of objects of very different scale.

The output of YOLO on a video is a set of categorized bounding boxes for each frame. Each bounding box is identified by a center location (x, y), width and height (w, h), and a confidence c, that is an estimate of $Pr(Class_i) \cdot IOU_{pred}^{truth}$, i.e., the probability of belonging to a given class weighted by the Intersection Over Union (IOU) of the ground truth and the prediction. We recall that the IOU between two bounding boxes is defined as the area of the intersection divided by the area of the union.

The choice of YOLOv3 is due to its being arguably the fastest object detection architectures with state-of-the-art performance on the major object detection benchmarks. In particular, we chose the YOLOv3 architecture with a 416×416 resolution input, that runs at 35 FPS on a Nvidia Titan X GPU. We opted not to use the higher-resolution version of the network (608×608), since it is significantly slower and the videos in our dataset have a resolution of 480×356. However, with higher resolution videos, we believe that using the more powerful network architecture could bring about an improvement in the detection, with a bottom line improvement of the performance of our overall algorithm, since the accuracy of the object detection sub-task is crucial in obtaining good results.

B. Dense optical flow

As customary in tasks that involve recognition of structure and action from the motion of objects across multiple frames, we compute a dense optical flow for each two consecutive frames to extract features that help in our classification task.

Optical flow can be broadly defined as the pattern of apparent motion of objects between consecutive frames due to the movement of the camera or the objects themselves. Given two consecutive frames at time $t-1$ and t, we compute a dense optical flow, that is a vector field $V(t)$ that contains, for each pixel of the first frame, a displacement vector $v \in \mathbf{R}^2$ that represents the movement of such pixel from the first frame to the second. We use Farnebäck's algorithm [16], as implemented in OpenCV. Our use of the dense optical flow is twofold:

- we directly exploit it to estimate the vanishing point of the image and construct the collision cone of the ego vehicle, as we will explain in more detail in the following paragraph;
- we extract from it a number of features that we use within the random forest classifier.

An example of dense optical flow can be seen, color-coded, in Figure 3.

C. Road scene pipeline

Once we have computed, for each frame t, a set of bounding boxes from the detection algorithm, and an optical flow field $V(t)$ from $t-1$ to t, we process the data with the aim of extracting the time-to-contact of the objects in each frame, and whether they are on a collision course with the ego vehicle. We proceed as follows:

1) We perform object tracking across frames starting from the YOLO detections. Multiple object tracking is a very challenging problem per se [17], and most advanced tracking methods are sophisticated and computationally heavy. We choose to adopt the following simple and efficient procedure, based on a bipartite graph matching formulation of the problem, where we greedily match the objects tracked so far with the detections in the successive frame. The algorithm is initialized by assigning a unique object id j to each detection i in the first frame with confidence $c_i \geq 0.6$. Then:

   for $t = 2$ to T **do**
   Compute a matching between the detections in frame t and the objects in frame $t-1$
   Assign to each matched detection i the object id j of its matched object
   Assign to each unmatched detection i with $c_i \geq 0.6$ a new unique id j

Discard all remaining detections in frame t
end for

The greedy matching iterates through the set of candidate pairs in order of IOU value between the tracked objects in frame $t-1$ and the detections in frame t, subject to the following constraints: each object in frame $t-1$ is matched with at most one detection in frame t; a detection can only be matched to an object of the same class; a match can only occur with a IOU greater than a threshold $\theta = 0.2$.

2) We apply a low-pass filter over time on the bounding boxes of each tracked object. This is necessary to reduce the imprecision due to the imperfect localization of the boxes by the object detector – we recall that the object detection algorithm is trained on single images, and runs on each frame independently.
3) We determine the time-to-contact (TTC) for each tracked object. The time-to-contact can be defined as $TTC = \frac{Z}{v}$, where Z is the actual distance of the object from the ego vehicle and v is the current speed, that is assumed to be locally constant. Both values are unknown, but exploiting the speed formulas derived in [18], given a pair of frames, the TTC can be expressed as

$$TTC = \frac{\Delta t}{s},$$

where Δt is the time difference between the two frames, and s is the scale change of the bounding box, that can be computed as:

$$s = \frac{w - w'}{w'}$$

being w and w' the width of the box in the two frames. The scale of change could be computed either with the height or the width of the boxes; indeed, in our implementation we use the smallest scale variation of the two dimensions, to reduce artifacts due to occlusions. The TTC is computed only on objects that have appeared for at least Δt seconds.
4) We use a RANSAC-based procedure to estimate the vanishing point as a robust average of the intersection points of randomly sampled optical flow vectors. Then, we compute the collision cone of the ego vehicle as the cone connecting two points at the base of the video with the vanishing point. This is done to estimate the part of the frame which is interested by the passage of the ego vehicle. The width of the base of the cone is tuned experimentally.
5) For each tracked object, we estimate if its trajectory will potentially collide with the ego vehicle based on a simple motion model. To do so, we compute a displacement vector at time t with respect to 15 frames in the past for each corner of the bounding box and we multiply these vectors by a scalar parameter which we have tuned experimentally. These vectors are used to translate the corners, thus the object, to a predicted future position. If the predicted bounding box enters the collision cone, we flag the object at time t as being on a *collision course* with the ego vehicle. This binary flag will then be used to extract high-level features for the classifier.

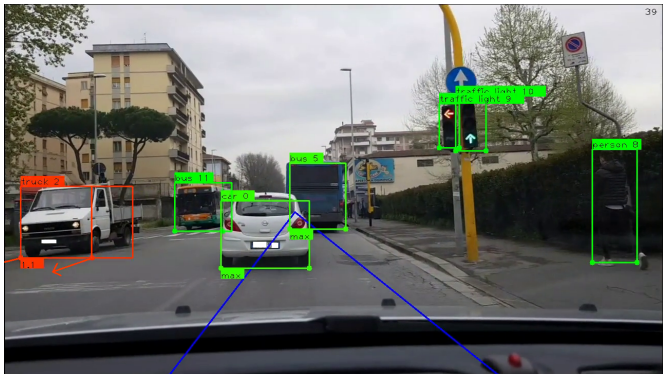

Fig. 4. Example of the output of the road scene pipeline.

The results of the procedure are a number of semantic annotations for each frame of the video. In Figure 4, we report an example of a frame automatically annotated with the output of our pipeline. All the objects displayed in the image have a progressive id assigned by the tracking procedure. The number on the bottom left of each box is the estimated time-to-contact (TTC), when available. For objects that are moving away, we show the value `max`. The TTC value is also encoded in the color of the bounding (green = high, red = small). The arrows on the bottom corners of a bounding box represent the trajectory estimated with our motion model. Note that the truck on the left has a relatively small TTC since it is nearing the ego vehicle, but is correctly marked as not being on a collision course (the trajectory arrows point outside of the collision cone).

For sake of compactness, we have omitted to describe in detail other minor corrections that are useful to enhance the accuracy of the TTC estimate and motion prediction. Among them, we mention a simple 3D correction procedure to estimate the face/rear of a vehicle based on its direction and supposed form factor. The result of the correction of a bounding box can be seen in Figure 4, applied to the truck on the left side of the image.

D. Feature extraction

After the road scene pipeline, we compute a number of features for the final classifier. We have two main groups of features extracted from the video, which are computed over the frames between the start and the end of the event.

One group includes features derived directly from the dense optical flow. To aggregate the full optical flow information, which consists of a 2-d vector for each of the 170 880 pixels, for each frame we compute the average horizontal and vertical optical flow in each cell of a 3×3 grid. Then, we aggregate these values over time, with the following statistical functions: `max`, `min`, `std`, `avg`.

Another feature we extract from the optical flow is one that helps detecting swerving manoeuvres, that are highly

correlated with near-crash events. To identify a swerve, which we define as a rapid turn towards a side followed by another to the opposite one, we filter the horizontal optical flow of the central cell using a convolution with a kernel that highlights a sequence of rapid changes of direction. Then, we aggregate the filtered signal across time with the statistical operators `max`, `std`, `avg`.

The second group of features includes high-level features derived from the road scene pipeline. The main products of the pipeline are the TTC and the detection of colliding objects. From this information, we extract a number of features related to the TTC value of objects on a collision course with the ego vehicle, as well as other statistics such as the average number of surrounding objects, or the maximum area of an object on a collision course. We also exploit the classifications obtained from the object detector, to construct different features for each class. As an example, we do not care about the TTC of a traffic light, and we do not include it as a feature, though we might be interested in knowing if a traffic light has been detected during the event. The feature we compute are all the combinations of the form: (`max`|`min`|`std`|`avg`) (TTC|Area) for (all|colliding) objects belonging to (all categories|all except traffic lights and stop signs|motorized vehicles|pedestrians and bicycles), and the average number of objects belonging to (all categories|traffic lights and stop signs|motorized vehicles|pedestrians and bicycles).

From the telematics data (speed and 3-axis acceleration), we extract statistical features over the time of the event with the operators `max`, `std`, `avg`, `min`.

Overall, we generate a set of almost 200 features. We are quite generous in the choice of features to include in the training, since we use them to train a Random Forest classifier, which is typically able to automatically select the most relevant features for the classification task. In Section IV, we use the trained Random Forest to provide an *a posteriori* ranking of the features in term of their usefulness for classification.

E. Random Forest classification

Finally, a Random Forest classifier [8] is trained on the output feature vectors for each event in the training set. We choose a Random Forest classifier for its well-known effectiveness and versatility on both binary and multi-category classification tasks, in addition to being extremely fast. We use the Random Forest implementation in scikit-learn [19], which uses Gini impurity to select the splits during the construction of the classification trees, and samples $\sqrt{m}$ features for each tree, where m is the total number of features. To control the model complexity, we limit the maximum tree depth and the number of trees in the forest. Moreover, to take into account the class imbalance, we weight the samples of the classes by the inverse of the class frequencies.

IV. Experimental results

We report experimental results for two scenarios. In the first experiment, we consider the binary problem of discriminating between dangerous and safe events, grouping together crash and near-crash events. In the second experiment, we extend the approach to the multi-category problem of classifying each event into one of the 3 classes: crash, near-crash, and safe. In both cases, we report results obtained in cross-validation rather than with a train/test split, due to the small number of crash events that we would rather not dilute. The folds of the cross-validation are the same in the two scenarios, and they are stratified by the 3 classes. All the experiments were run on an Intel Xeon CPU E5-2620 v4 @ 2.10GHz with 32GB of RAM and an Nvidia Tesla P100 GPU.

In the 2-class problems, we have 7035 dangerous events and 4082 safe events. Table I reports the confusion matrix of the 5-fold cross-validated results, obtained with a Random Forest with 2000 estimators and max depth 15.

TABLE I
Confusion matrix for the 2-class problem

		Predicted	
		Dangerous	Safe
Real	Dangerous	6592	443
	Safe	999	3083

The global accuracy is 0.87, and the balanced accuracy (average accuracy per class) is almost 0.85. The classifier achieves a good recall on the dangerous events (almost 0.94), and a precision of 0.87. However, the model is still classifying a significant number of safe events as dangerous ones: the recall of safe events is slightly above 0.75, while the precision is again around 0.87. We must remark that for a significant number of events the distinction between safe and dangerous is fuzzy even for a human subject. Indeed, in a small experiment we have run within our team, we estimated that a human reviewer would achieve a balanced accuracy of around 90%.

In the 3-class problem, we have 280 crashes, 6755 near-crash events, and 4082 safe events. The confusion matrix is reported in Table II. The global accuracy is more than 0.85, although the performance on the classes is not homogeneous. It appears that discriminating crashes and near-crashes is quite challenging; on crashes, by far the minority class, the model achieves a recall of 0.49, though with a high precision (more than 0.93). Of course, the small number of crashes does not help generalization. We have also analyzed the metadata included in the SHRP2 dataset to gain some insight on the kind of error made by our model on the crash events. Among the misclassified samples, around 50 contain an accident where the ego vehicle was hit from behind; in these cases, the video from a front-facing camera carries little information about the dynamics of the crash, and the acceleration profile might be similar to near-crash events. Another example of crash events that are especially challenging for our system (more than 10 errors) are animal strikes; these events typically unfold very quickly, and our current object detector is currently not trained to recognized animals such as deer or rabbits.

A ranking of feature importance can be obtained accumulating the improvement in Gini impurity obtained splitting on

TABLE II
CONFUSION MATRIX FOR THE 3-CLASS PROBLEM

		Predicted		
		Crash	Near-crash	Safe
Real	Crash	137	133	10
	Near-crash	9	6290	456
	Safe	1	980	3101

each feature during the training phase, as suggested in [20]. In both problems, the ranking of a model trained on the whole dataset shows that the most important feature (1st overall) to classify crash and near-crash from safe events is, perhaps unsurprisingly, the maximum longitudinal deceleration, i.e., the harshest brake during the event. Excluding that, the first features for importance are the area of the largest motorized vehicle on a collision course, and the value of the minimum TTC of any motorized object on a collision course. Indeed, we expected to find a close relation between the minimum TTC with the riskiness of a situation.

Among other highly ranked features are the maximum swerve value, that indicates presence of rapid evasive manoeuvres, and the standard deviation over time of the average optical flow in the central cells. Intuitively, this value correlates highly with harsh movements or strong vibrations.

Interestingly, the maximum speed (from telematics data) appears to be important to discriminate dangerous from safe events (top-10), while, in the 3-class problem, the *minimum* speed is also highly ranked. This might suggest that knowing whether the vehicle has stopped right after the event is important to classify it as a crash.

V. CONCLUSION

In this paper, we introduced an approach for the classification of crash and near-crash events, based on the integration of dashcam videos and telematics data collected from on-board sensors. In our system, we combine a state-of-the-art object detector algorithm with a number of other machine learning and traditional computer vision techniques.

Experimental results on a binary version of the problem, where we aim to discriminate dangerous events from safe ones, show that the global classification accuracy in cross-validation is over 87%. In a multi-category version of the problem, where we attempt to distinguish between crash, near-crash, and safe events, we obtain again a good global accuracy of over 85%, but with a weaker result on the less represented class.

In the near future, we envision our work will involve improving the object detection algorithm, for instance by training a model specifically on road scene datasets, such as those in [21], and enhancing the quality of the TTC estimate. We also believe that, for this and similar tasks, additional data with crash and near-crash events is necessary. This poses a challenge, due to their rare occurrence.

Possible directions for future research involve the extension to finer-grained classes, and the use of a combination of recurrent and convolutional neural networks to fully exploit sequential information. We believe the final goal of this line of research should be the development of an integrated event detection and classification system that can run with high accuracy in real-time on a stream of sensor data from multiple sources of information.

REFERENCES

[1] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, "Traffic monitoring and accident detection at intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 2, pp. 108–118, 2000.

[2] Y.-K. Ki and D.-Y. Lee, "A traffic accident recording and reporting model at intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 188–194, 2007.

[3] C.-Y. Chen, W. Choi, and M. Chandraker, "Atomic scenes for scalable traffic scene recognition in monocular videos," in *Proc. of the IEEE Winter Conference on Applications of Computer Vision*, 2016.

[4] F.-H. Chan, Y.-T. Chen, Y. Xiang, and M. Sun, "Anticipating accidents in dashcam videos," in *Asian Conference on Computer Vision*. Springer, 2016, pp. 136–153.

[5] J. White, C. Thompson, H. Turner, B. Dougherty, and D. C. Schmidt, "Wreckwatch: Automatic traffic accident detection and notification with smartphones," *Mobile Networks and Applications*, vol. 16, no. 3, 2011.

[6] C. Streiffer, R. Raghavendra, T. Benson, and M. Srivatsa, "Darnet: A deep learning solution for distracted driving detection," in *Proc. of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017.

[7] J. M. Hankey, M. A. Perez, and J. A. McClafferty, "Description of the SHRP 2 naturalistic database and the crash, near-crash, and baseline data sets," Virginia Tech Transportation Institute, Tech. Rep., 2016.

[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[10] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European Conference on Computer Vision*. Springer, 2014, pp. 740–755.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[14] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. of the International Conference on Machine Learning*, vol. 30, no. 1, 2013, p. 3.

[15] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[16] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandinavian Conference on Image Analysis*. Springer, 2003, pp. 363–370.

[17] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, and T.-K. Kim, "Multiple object tracking: A literature review," *arXiv preprint arXiv:1409.7618*, 2014.

[18] G. P. Stein, O. Mano, and A. Shashua, "Vision-based ACC with a single camera: bounds on range and range rate accuracy," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2003, pp. 120–125.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning*, 2nd ed. Springer, 2009.

[21] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.