Crash to Not Crash: Learn to Identify Dangerous Vehicles using a Simulator

Hoon Kim*, Kangwook Lee*, Gyeongjo Hwang & Changho Suh
School of Electrical Engineering
KAIST
Daejeon, South Korea
{gnsrla12,kw1jjang,hkj4276,chsuh}@kaist.ac.kr

Abstract

Developing a computer vision-based algorithm for identifying dangerous vehicles requires a large amount of labeled accident data, which is difficult to collect in the real world. To tackle this challenge, we first develop a synthetic data generator built on top of a driving simulator. We then observe that the synthetic labels that are generated based on simulation results are very noisy, resulting in poor classification performance. In order to improve the quality of synthetic labels, we propose a new label adaptation technique that first extracts internal states of vehicles from the underlying driving simulator, and then refines labels by predicting future paths of vehicles based on a well-studied motion model. Via real-data experiments, we show that our dangerous vehicle classifier can reduce the missed detection rate by at least 18.5% compared with those trained with real data when time-to-collision is between $1.6\,\mathrm{s}$ and $1.8\,\mathrm{s}$.

Introduction

Nearly 1.3 million people die in road crashes each year, and 3 out of 4 deaths are caused by incautious driving. One can save a significant number of lives by warning them ahead of a collision. The goal of collision prediction system (CPS) is to predict vehicle collisions as early as possible by using a variety of input sensors. While developing efficient CPS requires a large amount of labeled accident data, it is difficult to collect accident data in the real world, and it is even more difficult to accurately annotate accident data.

In this work, we tackle this challenge by developing a synthetic data generator built on top of a driving simulator. Specifically, we manipulate low-level internal functions of a video game called Grand Theft Auto 5 (GTA V) to synthesize accident and non-accident scenes. Each scene is labeled according to simulation results of nearby vehicles.

Potentially due to the difference between the driving model of AI agents in the simulator and that of human drivers, synthetic labels are very noisy, resulting in performance degradation when used for training. In order to improve the accuracy of synthetic labels, we propose a label adaptation technique, which fully leverages the flexibility of synthetic data generator. Our label adaptation algorithm first extracts the states of nearby vehicles by accessing internal variables of the underlying driving simulator. Using the extracted states, we refine labels by predicting vehicle paths as per a vehicle motion model.

Using the aforementioned approach, we generate a large synthetic accident dataset, and train efficient CNN-based classifiers with it. We evaluate the performance of our approach on a real-world accident dataset consisting of YouTube dashcam videos[1]. Our extensive experiments show that classifiers trained with our synthetic data better predict dangerous vehicles than those trained with real data. Specifically, when time-to-collision (TTC) is between $1.6\,\mathrm{s}$ and $1.8\,\mathrm{s}$, our approach achieves 18.5% lower missed detection rate than those trained with real data.

Related Works

Collision prediction algorithms: Due to the difficulty of collecting a large amount of accident dataset, most of the existing collision prediction algorithms are rule-based ones (Raut and Malik 2014). These approaches predict collisions by developing precise motion models, fitting the motions of the nearby vehicles, and estimating their future trajectories. For instance, one can compute the collision probability as a function of the distance to the nearby vehicles and the speed of ego-vehicle (Nakamori et al. 2002). However, these approaches typically assume availability of expensive devices (such as radar, LIDAR, or communication chips) to measure or receive the states of nearby vehicles.

To avoid the dependency on expensive hardwares, one may make use of low-cost cameras by developing a computer vision-based collision prediction systems. In a few recent works, the authors propose the use of convolutional neural networks (CNNs) for building efficient CPS (Wang and Kato 2017; Suzuki, Aoki, and Kataoka 2017; Chan et al. 2017). However, they could collect only a small amount of accident data due to data scarcity and difficulty of manual annotation. Different from the existing works, our synthetic data generator can produce a large enough dataset without requiring manual labeling. Further, existing computer vision-based algorithms classify dangerous scenes from safe scenes, and hence cannot capture which vehicle is the source of danger. On the other hand, our proposed framework first

*These two authors contributed equally

[1] https://sites.google.com/view/crash-to-not-crash

detects/tracks vehicles in a scene, and then classifies dangerous ones from safe ones, improving interpretability.

Synthetic dataset via simulators: The idea of using simulators or video games to generate an arbitrarily large amount of synthetic data with labels has been shown useful for various tasks such as semantic segmentation (Richter et al. 2016) and eye gaze estimation (Shrivastava et al. 2017; Lee, Kim, and Suh 2018). It voids the need of costly annotation procedure: For instance, annotating a single image with semantic segmentations takes about 60 minutes but it takes almost no time with a photo-realistic simulator (Richter et al. 2016). Moreover, it even enables more detailed annotation, which is traditionally considered impossible: One may annotate each vehicle in a driving scene with its velocity, acceleration, wheel angle, and vehicle orientation. In this work, by manipulating driving agents in the virtual world, we freely generate dangerous driving scenes, which are not observed frequently in the real world.

Domain adaptation: Even though modern simulators and video games provide photo-realistic images, there still exists a gap between synthetic and real data distributions. To minimize this gap, one may adapt algorithms trained with synthetic data to real data via *domain adaptation*. One approach is to use both real and synthetic data when training (Richter et al. 2016). Another promising line of approaches is based on Generative Adversarial Networks (GANs). Particularly, GAN-based unsupervised image-to-image translation algorithms have been shown useful for adapting a model trained on a labeled data to another *unlabeled* dataset. This task, called unsupervised domain adaptation, is useful particularly for our setting since it is much easier to collect unlabeled accident images than to collect labeled ones. The first GAN-based domain adaptation algorithm is proposed in (Shrivastava et al. 2017) and (Bousmalis et al. 2017). They first learn a forward mapping from synthetic images to real images via GAN, translate labeled synthetic images to real-like images, and then train a model with the translated images. They achieve the state-of-the-art performance on some unsupervised domain adaptation tasks using this approach. In (Lee, Kim, and Suh 2018), the authors take a backward approach: They map real images to the synthetic domain at inference time and then apply a model that is trained in the synthetic domain.

In order to generate synthetic labels, one can simply simulate the dynamics of vehicles and see whether or not any of the nearby vehicle is colliding with the ego-vehicle in near future. Due to the difference between the driving model of AI agents in the simulator and that of human drivers, such a labeling method does not provide accurate labels, resulting in significant performance degradations (See the section on domain adaptation for more details). To tackle this challenge, we make a novel use of collision risk assessment algorithms. In short, we extract hidden states of vehicles from the simulator, and assess each vehicle's collision risk via a motion model. As a result, we achieve the best classification performance by adapting both features and labels.

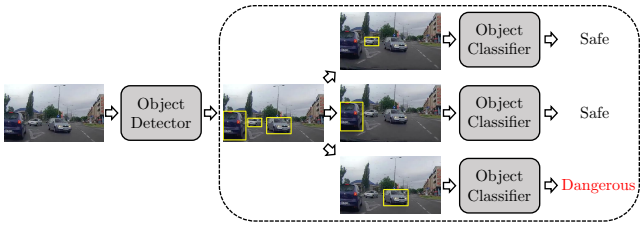


Figure 1: A two-stage decomposition of collision prediction. The first stage detects/tracks vehicles in the input image(s), and the second stage classifies dangerous vehicles. Our goal is to design an accurate and efficient classifier.

Self-driving and collision avoidance: Another line of related works is about self-driving and collision avoidance algorithms. In (Pomerleau 1989), the authors demonstrate that a successfully trained neural network can mimic the way humans drive. Recently, deep learning has been applied to self-driving (Chen et al. 2015; Bojarski et al. 2016), achieving extraordinary performances. In (Bojarski et al. 2017), the authors explain how their deep neural networks perceive the scene, and show that they pay more attentions to key objects such as lane markings, nearby vehicles, etc. In (Kahn et al. 2017; Thorsson and Steinert 2016), the authors apply reinforcement learning to develop collision avoidance systems. Despite of the huge success, these approaches cannot efficiently predict and handle dangerous scenarios, which are rarely observed during training. Thus, our collision prediction algorithms can provide an additional level of safety to self-driving and collision avoidance systems.

Problem Formulation and Datasets

Collision prediction problem

A collision prediction algorithm can be divided into two phases as shown in Fig. 1. First, it detects and tracks every vehicle in a sequence of images. Since object detection/tracking algorithm does not need to be trained with accident data, one can easily make use of an off-the-shelf object detection/tracking algorithm (Ren et al. 2015; Dai et al. 2016; Redmon and Farhadi 2017; Liu et al. 2016). The second phase, which is of our main interest, takes the input images together with the bounding boxes of a vehicle, and classifies *whether or not the vehicle indicated by the bounding boxes is dangerous*. This procedure is applied to each vehicle to identify all dangerous vehicles from the input scene.

The dangerous vehicle classification problem can be formally defined as follows. A natural definition of *dangerous vehicles* is probabilistic: A vehicle is *dangerous* at time T if it will collide with the ego-vehicle, in time $t \in [T, T + T_{\text{look-ahead}}]$ with probability larger than some threshold, say ϵ. Each labeled sample consists of a feature $X = (X^{T-\delta}, X^{T-2\delta}, \ldots, X^{T-d\delta})$ and a binary label $Y \in \{0, 1\}$. Here, X^t denotes the frontal view taken by the ego-vehicle at time t and the bounding box of the vehicle of interest at that time. Assuming the frontal view is given as an RGB image of size $w \times h$ and encoding the bounding box of a vehicle as a binary image of the same size, one can treat

Figure 2: Sample images from YouTubeCrash: Accident data (first row) and non-accident data (second row)

the bounding box as the fourth channel, i.e., $X^t \in \mathbb{R}^{w \times h \times 4}$.

Given the train data $[(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)]$, our goal is to find a mapping $h_\theta(X) := p(Y = 1|X; \theta)$ that maximizes the log-likelihood of the dataset, i.e., we solve the following maximum likelihood problem:

$$\max_\theta \sum_{i=1}^{N} Y_i \log h_\theta(X_i) + (1 - Y_i) \log (1 - h_\theta(X_i)). \quad (1)$$

In this work, we will consider a class of classifiers that are parameterized by weights θ.

While we take the log-likelihood as an objective function to maximize, one may consider more useful metric such as the area under curve (AUC) of the receiver operating characteristic (ROC) curve, or AUC in short. AUC is particularly useful when the number of positive samples largely differ from that of negative samples. In our application, the number of safe vehicles (negative samples) is much larger than that of dangerous vehicles (positive samples), and hence we choose AUC as the primary metric of this work. We also report missed detection rate at a fixed false alarm rate.

YouTubeCrash

For evaluation of dangerous vehicle classification algorithms, we prepare a real dataset consisting of 122 dashcam videos collected from YouTube (CarCrashesTime 2014), which we call YouTubeCrash. These dashcam videos include diverse accident scenarios such as abrupt lane changes, sudden stops, and signal violations. From each video, we extract a pair of accident scene and non-accident scene: A clip of length 2.0 s just before the accident is defined as an accident scene, and another clip of the same length starting from the beginning of the video is defined as a non-accident scene. When an accident clip and the corresponding non-accident clip overlaps with each other, we discard the non-accident clip. As a result, we collected 122 accident and 100 non-accident scenes in total. We randomly pick 61 accident and 50 non-accident scenes to form train set, and the rest of the dataset form test set. Some statistics of YouTubeCrash are shown in Table. 1.

To generate samples from each scene, we perform the following preprocessing. For the vehicle that crashes with the ego-vehicle at time T, we generate the number $T_{\text{look-ahead}}/\delta$ of samples as follows:

$$\begin{aligned} &((X^{T-\delta}, X^{T-2\delta}, \ldots, X^{T-d\delta}), 1), \quad (2)\\ &((X^{T-2\delta}, X^{T-3\delta}, \ldots, X^{T-(d+1)\delta}), 1), \ldots,\\ &((X^{T-T_{\text{look-ahead}}}, X^{T-T_{\text{look-ahead}}-\delta}, \ldots, X^{T-T_{\text{look-ahead}}-(d-1)\delta}, 1), \end{aligned}$$

Table 1: Statistics of YouTubeCrash and GTACrash

Number of	YouTubeCrash	GTACrash
Accident scenes	122	7720
Non-accident scenes	100	3661
Positive samples	2096	128347
Negative samples	11486	623173

where $T_{\text{look-ahead}}$ denotes the maximum look-ahead time for accident prediction, and δ is the time between consecutive frames. In this work, we set $\delta = 0.1$ s, $T_{\text{look-ahead}} = 18\delta = 1.8$ s, and $d = 3$. That is, each sample consists of 3 consecutive frames, and the maximum look-ahead time is 18 frames in future. Further, each scene contains $T_{\text{look-ahead}}/\delta = 18$ samples per vehicle.[2] Recall that X^t denotes the RGB image and the bounding box of the dangerous vehicle at time t. Each bounding box of the vehicle is encoded as a binary map of the same dimensionality as the RGB channels. We manually annotate each sample with bounding boxes using LabelImg (Tzutalin 2015), an open-source software for manual bounding-box annotation. Shown in Fig. 2 are some samples from YouTubeCrash: See Supplementary Materials for more samples. Red bounding boxes denote dangerous vehicles while green ones denote safe ones.

For all the other vehicles (in an accident scene) that are not going to crash with the ego-vehicle, we generate samples in a similar way but with negative labels. Non-accident scene are used to generate negative samples: Per each vehicle in a non-accident scene, we generate negative samples.

Synthetic Data Collection

GTACrash

In this section, we describe our data generation framework and introduce our dataset, which we dub as GTACrash. Our synthetic dataset GTACrash is collected from a popular video game named Grand Theft Auto V (GTA V). We now detail how we collect our synthetic dataset.

In order to collect a large amount of driving data, we first implement our own data generation framework based on Script Hook V, an open-source library that enables access to the low-level internal functions of GTA V.

A few important functions available in our framework, which play key roles in our data generation

[2] To be more precise, each scene may contain *less than* or equal to 18 samples per vehicle since some samples are removed if the dangerous vehicle is invisible (due to occlusion or so) in the scene.

process, are described as follows: `Start_Cruising(s)` lets the player's car cruise along the lane at speed `s`; `Get_Nearby_Vehicles()` returns the list of vehicles in the current scene of the game; `Set_Vehicle_Out_Of_Control(v)` makes vehicle `v` drive out of control; `Is_My_Car_Colliding()` checks whether the ego-vehicle is colliding with some other vehicle; `Get_Bounding_Box(v)` returns the bounding box of vehicle `v`; `Get_Internal_States(v)` returns the (x, y) coordinates of the vehicle, vehicle orientation, velocity, acceleration, and yaw rate (rotation rate around z-axis) of vehicle `v`; and `Set_Wheel_Angle(v, a)` sets the wheel angle of the specified vehicle `v` as `a`.

A data generation process begins with calling `Start_Cruising(s)`, making the player's car cruise along the lane at the specified speed. The player keeps exploring the virtual world until the end of the generation process. This is possible because the player's car has a full access to the internal states of the virtual world such as the road lanes and the map of the world. We now describe two different data generation modes: one for generating non-accident scenes and the other for accident ones. In both modes, the data generator collects screenshots of the game at the rate of 10 frames per second, i.e., $\delta = 0.1\,\mathrm{s}$, and keeps only a subset of the collected screenshots according to certain rules.

We first illustrate how we collect driving scenes. The data collector samples random times according to a Poisson process. For each of the sampled time, the collector enters the normal driving mode with probability half or the insane driving mode with probability half.

Under the insane driving mode, one of the nearby vehicles is chosen uniformly at random, and it will start driving out of control. We implement this by utilizing the functions available in our data generation framework as follows. We call `Get_Nearby_Vehicles()` to retrieve the list of nearby vehicles. We then choose one of those nearby vehicles at random, and make it start driving carelessly by calling `Set_Vehicle_Out_Of_Control(v)`. We observe that with high probability, the chosen car eventually ends up with crashing into the player's car. Since the player continues driving at the constant speed following the lane without attempting to avoid the collision, one can obtain risky scenes in this way. To check whether or not a car accident actually happens, we make use of `Is_My_Car_Colliding()`. When an accident is detected at time T, we take the 20 screenshots of the game screen at time $T-\delta, T-2\delta, \ldots, T-20\delta$. To make screenshots more realistic, we randomly tilt the virtual dashcam by a small angle and then captures screenshots. This collection of 20 screenshots is called an accident scene. Similar to `YouTubeCrash`, each 3 consecutive frames form a sample, and hence 18 samples are generated for each vehicle in an accident scene. For the vehicle that collides with the ego-vehicle, we label its samples with 1. For all the other vehicles, their samples are labeled with 0. We also obtain each vehicle's bounding box via `Get_Bounding_Box(v)`, encode it as a binary image, and include it in the corresponding sample.

Under the normal driving mode, all the vehicles in the simulator drive normally according to the original driving rule of GTA V. Data samples are generated in a similar way, and the only distinction is that samples are always labeled with 0. Scenes, frames, and samples generated under this mode are called non-accident.

Fig. 3 shows some sample driving images from `GTACrash`. Shown in the first row are consecutive frames from an accident scene, and shown in the second row are those from a non-accident scene. See Supplementary Materials for more samples.

Overview of `GTACrash`: Following the above procedure, we collect synthetic data for 72 hours on a single computer running GTA V. As shown in Table. 1, our dataset `GTACrash` consists of 3661 non-accident scenes and 7720 accident scenes. The total number of positive samples is 128347, and the total number of negative samples is 623173. Note that the dangerous vehicle in the first row is marked with red bounding boxes while all the others are marked with green bounding boxes. Further, the dataset is highly diverse: The images are taken in arterial roads and highways, in days and nights, and on sunny, rainy, and snowy days.

We note that the distribution of our synthetic data is different from that of real data since our current simulator cannot capture all possible accident patterns that happen in reality. For instance, our dataset does not include sudden stop accidents, which frequently happen in the real world. Regardless of the bias in the `GTACrash`, our goal is to demonstrate that the model trained with biased but large synthetic data can outperform that trained with unbiased but small real data.

Algorithms

In this section, we describe our algorithms along with its variations and baseline approaches.

Classification algorithms (RGB + B.Box)

We first illustrate our approach that takes both images and bounding boxes and outputs the probability of a vehicle being dangerous. Recall that each sample's feature is a sequence of d frames, say $(X^1, X^2, \ldots, X^d)$, and each frame is encoded as a 4-channel (RGB and Bounding box) image. We design a neural network that takes a concatenation of d frames as input. That is, we concatenate $X^1, X^2, \ldots, X^d$ to obtain $X^{1:d} \in \mathbb{R}^{w \times h \times 4d}$, and then feed it into a smaller version of the VGG16 network (Simonyan and Zisserman 2014). Specifically, we modify the original VGG16 network by replacing the FC (fully-connected) layers of the original architecture (the layers after pool5) with three FC layers. The first and second hidden layer is of size 100, and the last layer of size 2 corresponds to the binary classes. We train our model using AdamOptimizer with mini-batches of size 32 and momentum parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ (Kingma and Ba 2015). We use 10^{-4} as the initial learning rate, and decay it by a factor of 10 at the end of every epoch.

Figure 3: Sample images from GTACrash: Accident data (first row) and non-accident data (second row)

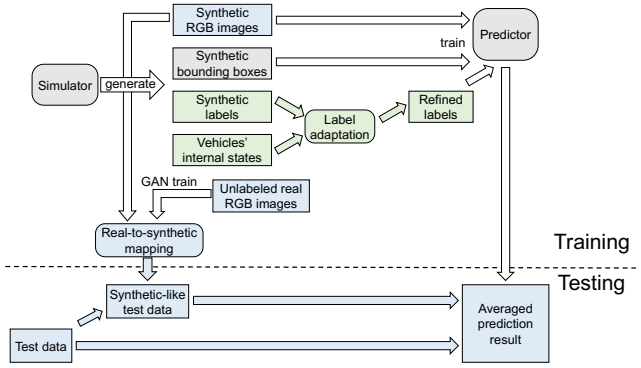


Figure 4: Illustration of our domain adaptation algorithm

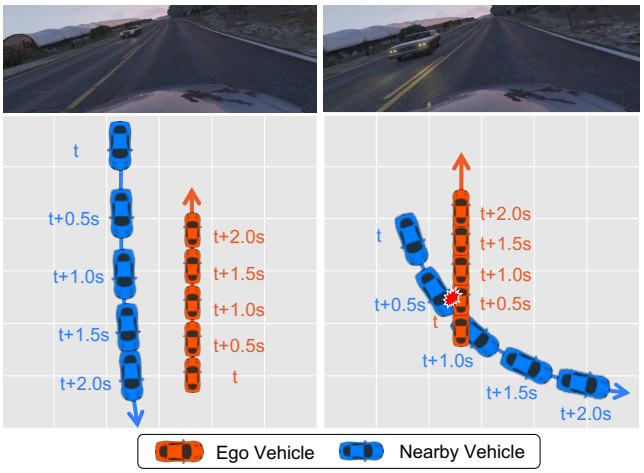


Figure 5: A sample positive scene and predicted paths: The nearby vehicle on the left hand side is not going to collide with the ego-vehicle according to model-based path prediction, and our label adaptation algorithm will flip the label of this sample. On the other hand, the nearby vehicle is going to collide with the ego-vehicle according to path prediction, so we keep its label as positive.

Baseline algorithms and limitations

We now illustrate two simple baseline algorithms. **(B.Box-size)** The first one classifies vehicles based on the sizes of their bounding boxes only. Specifically, we train a 5-layer fully-connected neural network (d-10-10-10-10-2), where the input layer is a d-dimensional vector, whose entries are the size of the bounding boxes in d consecutive frame. This is a promising approach if the goal is to predict imminent collisions since the dangerous car (and its bounding box) must be much larger than other safe vehicles (and their bounding boxes). However, if the look-ahead time gets longer, such a size-based predictor is expected to perform worse than more sophisticated algorithms since non-imminent accidents might not be predictable without examining subtle visual clues. **(B.Box)** Another baseline algorithm is similar to our proposed algorithm but it only takes the bounding box of the input, ignoring its RGB channels. While this is expected to outperform "B.Box-size" algorithm, it is also limited since it cannot distinguish between two different inputs with the same bounding boxes. For instance, it cannot tell difference between a vehicle turning toward the ego-vehicle from another one turning toward the opposite direction if their bounding boxes are the same.

Domain adaptation

We now describe our domain adaptation algorithm, illustrated in Fig. 4. Our domain adaptation algorithm can be viewed as a combination of forward-mapping approach (Shrivastava et al. 2017) and backward-mapping approach (Lee, Kim, and Suh 2018): It refines labels at training time (forward) while it refines features at inference time (backward).

We first explain why one can achieve improved performance by adapting labels in our application. Recall that our synthetic data generator labels a sample with 1 if the corresponding vehicle collides with the ego-vehicle according to simulated paths of them. If the driving algorithm of the underlying video game (GTA V) is identical or close enough to that of human drivers, such simulated paths are distributed similarly to those of the real world, and hence synthetic labels will be distributed identically to real labels. However, we observe that this is not the case in our application, and synthetic labels are biased. This is because the driving algorithms implemented in our simulator are simple rule-based ones that cannot fully capture complicated mechanisms of human drivers. This makes a non-negligible fraction of synthetic labels inconsistent with real labels, i.e., what human drivers would expect from driving scenes. Hence, one needs to apply a domain adaptation algorithm for labels.

To see this, consider an accident scene in GTACrash shown in Fig. 5. In these images, there exists a single visible vehicle near the ego-vehicle, and this vehicle labeled as positive (dangerous) since its simulated future path collides with that of the ego-vehicle. However, on the left hand side, it does not seem dangerous at all since it seems to safely follow

its lane without any abnormal behaviors. On the right hand side, the vehicle seems dangerous since its wheels are facing toward the ego-vehicle. This implies that the path simulated by the data generator is not always consistent with the path predicted by humans, and we need to refine such labels.

To resolve this, we propose a simple yet novel label adaptation method. Specifically, we adopt a path prediction model called the *Constant Turn Rate and Acceleration (CTRA)* model (Schubert, Richter, and Wanielik 2008). Even though the CTRA model is a simple curvilinear model that cannot capture the correlation between the velocity and the yaw rate (rotation rate around the z-axis), it is shown to achieve the state-of-the-art path prediction performance in the real world when various sensor measurements are available. More specifically, assuming constant turn rate ω and acceleration a, it predicts a path according to the following state transition equation:

$$\begin{aligned}&(x(t+T), y(t+T), \theta(t+T), v(t+T)) \\ &= (x(t), y(t), \theta(t), v(t)) + (\Delta x(T), \Delta y(T), \omega T, aT),\end{aligned} \tag{3}$$

where x, y, θ, v denote x-coordinate, y-coordinate, vehicle orientation along the z-axis, and velocity of the vehicle, respectively, and

$$\begin{aligned}\Delta x(T) = \omega^{-2}[&(v(t)\omega + a\omega T)\sin(\theta(t) + \omega T) \\ &+ a\cos(\theta(t) + \omega T) - v(t)\omega \sin\theta(t) - a\cos\theta(t)],\end{aligned} \tag{4}$$

$$\begin{aligned}\Delta y(T) = \omega^{-2}[&(-v(t)\omega - a\omega T)\cos(\theta(t) + \omega T) \\ &+ a\cos(\theta(t) + \omega T) + v(t)\omega \cos\theta(t) - a\sin\theta(t)].\end{aligned} \tag{5}$$

Thus, as long as one has an access to the state vector, turn rate, and acceleration of a vehicle, it can predict a path of the vehicle using the state transition equations. However, such information is not available in real dataset, and also it is nearly impossible to manually annotate it. On the other hand, such information can be made available in synthetic dataset by extracting the internal states from the underlying driving simulator. Specifically, one can first call `Get_Nearby_Vehicles()` to retrieve the list of nearby vehicles. Then, one can call `Get_Internal_States(v)` for each vehicle to retrieve $(x(t), y(t), \theta(t), v(t))$ and (ω, a) of the vehicle. By repeating this, we can compute the predicted path of each vehicle as per the CTRA model, and assign a positive label if and only if the predicted path intersects with that of the ego-vehicle. Once the new labels are obtained, we replace the synthetic labels of `GTACrash` with those obtained via our label adaptation algorithm.

As an example, let us revisit the positive samples shown in Fig. 5. We extract internal states of the ego-vehicle and the nearby vehicle, and predict its path as per the CTRA model. Visualized in the second row are predicted paths of them. According to them, the first image does not seem dangerous, hence its refined label is 0. Our forward adaptation method for labels is illustrated in green parts in Fig. 4.

For RGB images, we employ the backward translation approach (Lee, Kim, and Suh 2018). That is, we first train a classifier using the original synthetic images, and then at inference time we map input images to the synthetic domain before we feed them into the classifier. To achieve this, we learn a bidirectional mapping via CycleGAN with feature loss between `GTACrash` and an independent set of YouTube dashcam videos, which do not necessarily contain car accidents. We also tried learning a mapping between `GTACrash` and `KITTI` dataset (Geiger, Lenz, and Urtasun 2012), which is one of the most popular dataset of driving images, but did not observe improvements in performance. Note that a backward mapping learned from data is known to output highly distorted images in some cases. To mitigate this, we apply the classifier to both the original input and the refined input, and take the average output as the final prediction result. Our backward domain adaptation method for features is illustrated in blue parts in Fig. 4.

Table 2: Test AUC of classification algorithms on `YouTubeCrash`. Column titles denote which training dataset is used for each configuration.

Alg\Training data	`YouTubeCrash`	`GTACrash`
5-layer NN (B.Box-size)	**0.8766**	0.8812
CNN (B.Box)	0.8708	0.8992
CNN (RGB + B.Box)	0.8730	**0.9086**

Table 3: Test AUC of CNN (RGB + B.Box) with and without feature/label adaptation.

	w/o label adapt.	w/ label adapt.
w/o feature adapt.	0.9086	0.9154
w/ feature adapt.	0.9102	**0.9164**

Experimental Results

Comparison results

We now evaluate the performance of our algorithm and compare it with those trained with real data. Further, to observe the efficacy of synthetic data, we train each classifier once with `YouTubeCrash` and once with `GTACrash`, and compare their performances on `YouTubeCrash` test set. Shown in Table 2 are the comparison results. When trained with `YouTubeCrash`, the best performance is achieved by the simplest classifier 'B.Box-size'. This is due to overfitting since the model complexity of the other classifiers is too large considering the small training dataset.

On the other hand, when trained with `GTACrash`, the best performance is achieved by our proposed classifier (RGB + B.Box). This demonstrates the usefulness of synthetic data for training dangerous vehicle classifier since it enables training of highly complex models that take into account of additional source of information.

We also observe that it is beneficial to use all of the three frames in a sample to achieve the best performance: See Supplemental Materials for more details.

Domain adaptation

To further improve the classification performance, we apply our domain adaptation algorithms for features and/or

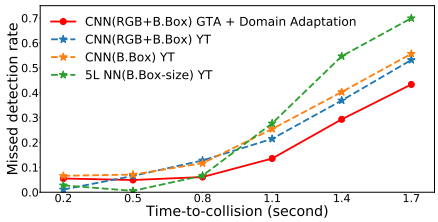


Figure 6: Missed detection rate vs time-to-collision. GTA: trained w/ GTACrash; YT: trained w/ YouTubeCrash.

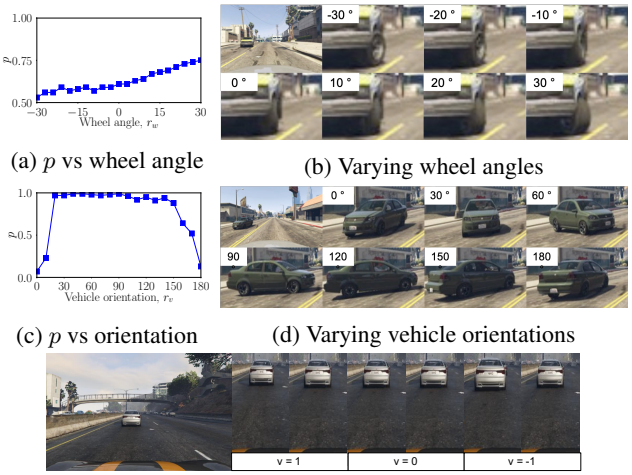
labels. In Table 3, we report the performance of our proposed classifier (RGB + B.Box) with and without domain adaptation, and refer the readers to Supplemental Materials for similar comparison results for additional experimental results. We observe that the classification performance improves when we apply either label adaptation or feature adaptation. The best performance is achieved when we apply both adaptation algorithms. Compared with the best performance achieved with YouTubeCrash, AUC is increased by 4.5%. We make a remark that even though our approach is trained only with our synthetic dataset GTACrash, it can easily match or outperform the best performance achieved with YouTubeCrash. We refer the readers to Supplementary Materials for a demo video.

Missed detection vs time-to-collision

We now observe missed detection rates as a function of TTC. We first set the classification threshold such that all algorithms have the same false alarm rate of 0.15. We then group samples into bins of length $0.3\,\mathrm{s}$ according to their TTC. For instance, the first group is for TTC between $0.1\,\mathrm{s}$ and $0.3\,\mathrm{s}$, and the last group is for TTC between $1.6\,\mathrm{s}$ and $1.8\,\mathrm{s}$. We then measure the missed detection rate of each group.

Plotted in Fig. 6 are the experimental results. Our approach, plotted as a red curve, achieves the best performance when TTC is larger than or equal to $0.7\,\mathrm{s}$. When TTC is between $1.6\,\mathrm{s}$ and $1.8\,\mathrm{s}$, our approach reduces the missed detection rate by 18.5%. The blue curve corresponds to the performance of CNN trained with real data, i.e., the approach proposed in (Wang and Kato 2017). We also train other baseline algorithms (5-layer NN with B.Box-size and CNN with B.Box) with real data. One can see that the performance of B.Box size-based classifier is the best when TTC is less than $0.8\,\mathrm{s}$, implying that very imminent accidents can be predicted simply by inspecting the size of bounding boxes.

Let us elaborate on why our proposed model significantly outperforms other simple baselines when TTC is large and why other simple baselines are better when TTC is small. When TTC is small, dangerous vehicles are closely located to the ego-vehicle, so it is easy to predict accidents based on bounding boxes. Thus, in this regime, simple baselines such as CNN with B.Box and 5-layer NN with B.Box-size would suffice to achieve a good performance while our model could suffer from overfitting. When TTC is large, a complex model is necessary for identifying distant dangerous vehicles.



(a) p vs wheel angle

(b) Varying wheel angles

(c) p vs orientation

(d) Varying vehicle orientations

(e) Varying velocities

Figure 7: Key visual factors of dangerous vehicles

Learning key visual factors

In the previous sections, we observed a non-negligible performance gap between CNN (RGB + B.Box) and CNN (B.Box). This is because CNN (B.Box) cannot distinguish between two images with different appearances but with the same bounding boxes. A natural question then arises: What are the subtle visual factors that help classify dangerous vehicle? By examining our CNN (RGB + B.Box) classifier optimized on GTACrash, we identify the following key visual factors of dangerous cars.

- (wheel angle) As wheels face toward the ego-vehicle, the accident probability increases (Fig. 7a, 7b).
- (vehicle orientation) As a vehicle faces toward the central line, the accident probability increases (Fig. 7c, 7d).
- (velocity) As a vehicles moves further away, the accident probability remains zero. As a vehicle moves toward the ego-vehicle, the higher its velocity is, the higher the accident probability (Fig. 7e).

We refer the readers to Supplemental Materials for detailed description of experimental setups.

Different CNN architecture

We remark that we simply chose VGG16 as our main CNN model since we observed similar experimental results with various CNN approaches. To verify this, we substituted the base CNN of our proposed algorithm (CNN with RGB+B.Box) and the baseline algorithm (CNN with B.Box) with Resnet50. Our proposed algorithm trained with GTACrash achieved test AUC of 0.9097, whereas the best test AUC among the algorithms trained with YoutubeCrash is 0.8776 by CNN with B.Box. This demonstrates the efficacy of our synthetic dataset, regardless of the base CNN model. Furthermore, with our proposed domain adaptation method, we could improve the performance from 0.9097 to 0.9241, demonstrating that our adaptation method works regardless of the CNN architectures.

Conclusion

In order to develop a computer vision-based algorithm for dangerous vehicle classification, we develop a synthetic data generator and propose a novel domain adaptation algorithm for labels. We show that our classifier trained with synthetic data outperforms those trained with real data.

We conclude the paper by presenting future directions. The distribution of accident patterns implemented in our simulator is still far from that of real accidents, and one interesting open question is how this gap can be reduced. One may also collect an even richer dataset that includes stereo camera inputs, rear/side camera inputs and audio input.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2018R1A1A1A05022889).

References

Bojarski, M.; Testa, D. D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; Zhang, X.; Zhao, J.; and Zieba, K. 2016. End to end learning for self-driving cars. *CoRR* arXiv:1604.07316.

Bojarski, M.; Yeres, P.; Choromanaska, A.; Choromanski, K.; Firner, B.; Jackel, L.; and Muller, U. 2017. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR* arXiv:1704.07911.

Bousmalis, K.; Silberman, N.; Dohan, D.; Erhan, D.; and Krishnan, D. 2017. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

CarCrashesTime. 2014. Car Crashes Time. https://www.youtube.com/user/CarCrashesTime. Accessed: 2018-09-01.

Chan, F.-H.; Chen, Y.-T.; Xiang, Y.; and Sun, M. 2017. Anticipating accidents in dashcam videos. In *Computer Vision – ACCV 2016*, 136–153.

Chen, C.; Seff, A.; Kornhauser, A.; and Xiao, J. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2722–2730.

Dai, J.; Li, Y.; He, K.; and Sun, J. 2016. R-FCN: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems 29*. 379–387.

Geiger, A.; Lenz, P.; and Urtasun, R. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3354–3361.

Kahn, G.; Villaflor, A.; Pong, V.; Abbeel, P.; and Levine, S. 2017. Uncertainty-aware reinforcement learning for collision avoidance. *CoRR* arXiv:1702.01182.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *2015 International Conference on Learning Representations (ICLR)*.

Lee, K.; Kim, H.; and Suh, C. 2018. Simulated+unsupervised learning with adaptive data generation and bidirectional mappings. In *2018 International Conference on Learning Representations (ICLR)*.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. SSD: Single shot multibox detector. In *Computer Vision – ECCV 2016*, 21–37.

Nakamori, T.; Iwasa, M.; Ishikawa, N.; and Nakajima, M. 2002. Vehicle front scene watching from the sequence of road images. In *Applications of Digital Image Processing XXV*, volume 4790, 425–433.

Pomerleau, D. A. 1989. ALVINN: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*. 305–313.

Raut, S. B., and Malik, L. G. 2014. Survey on vehicle collision prediction in VANET. In *2014 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 1–5.

Redmon, J., and Farhadi, A. 2017. YOLO9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517–6525.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28*. 91–99.

Richter, S. R.; Vineet, V.; Roth, S.; and Koltun, V. 2016. Playing for data: Ground truth from computer games. In *Computer Vision – ECCV 2016*, 102–118.

Schubert, R.; Richter, E.; and Wanielik, G. 2008. Comparison and evaluation of advanced motion models for vehicle tracking. In *2008 11th International Conference on Information Fusion*, 1–6.

Shrivastava, A.; Pfister, T.; Tuzel, O.; Susskind, J.; Wang, W.; and Webb, R. 2017. Learning from simulated and unsupervised images through adversarial training. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2242–2251.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR* arXiv:1409.1556.

Suzuki, T.; Aoki, Y.; and Kataoka, H. 2017. Pedestrian near-miss analysis on vehicle-mounted driving recorders. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, 416–419.

Thorsson, J. L., and Steinert, O. 2016. Neural networks for collision avoidance. Master's thesis, Chalmers University of Technology.

Tzutalin. 2015. LabelImg. https://github.com/tzutalin/labelImg. Accessed: 2018-09-01.

Wang, Y., and Kato, J. 2017. Collision risk rating of traffic scene from dashboard cameras. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 1–6.