Predicting Vehicle Crashes from Dashboard Camera Footage Using Both a Linear Approximation and Machine Learning-Based Approach

Suchet Sapre

TJHSST Computer Systems Research Lab 2020

Dr. Gabor, Period 3

Abstract (187 words)

Vehicular crashes are one of the leading problems within society. According to the National Highway Traffic Safety Administration, 102 car crashes occurred per day during 2016 in the US alone. Additionally, as we begin approaching the age of commercial self-driving vehicles, there is an ever-growing concern for the interaction between self-driving vehicles and human drivers. Namely, these self-driving vehicles need a better understanding of their environment in order to better anticipate human errors. In this study, I utilized a dashboard camera (dashcam) video dataset created by the VSLab based in Taiwan to develop an automated vehicle crash prediction algorithm. I implemented two approaches: linear approximation approach which compared vehicles' theoretical trajectories to predict whether or not a crash would occur (Naive Crash Prediction algorithm) and a Machine Learning (ML) approach which trained a deep neural network (DNN)-based model, to identify dashcam videos that contained crashes. Here we show that using raw dashcam videos to train a Convolutional Neural Network (CNN) displayed a high performance in predicting vehicle crashes. The best result this method obtained was predicting a vehicle crash 28 frames or 1.12 seconds ahead.

Introduction

Vehicle crashes are an extremely prevalent problem in today's society. With the rapid growth of technology, getting distracted while driving is becoming very easy. Although the number of car accident deaths has been decreasing throughout the past decades as depicted in Figure 1, over 4.4 million people were still injured in vehicle accidents in this US in 2019 alone [2]. Advanced driver attention monitoring systems and newer seatbelt technology are helping, but clearly a more effective solution is necessary to make a significant impact on car crash numbers.

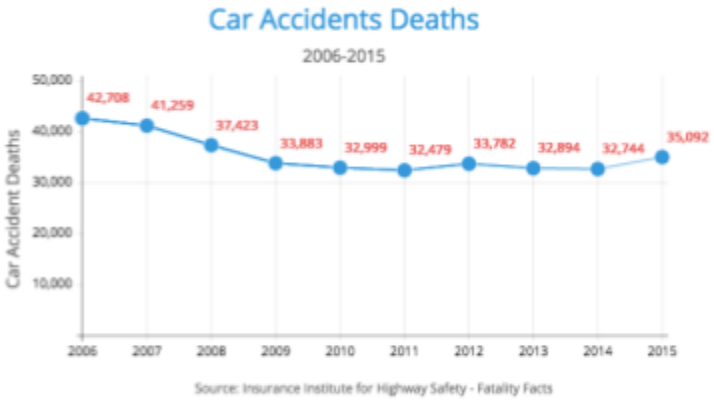

Figure 1. Car Accident Deaths vs. Time Graph

Another important issue to consider is the interaction between human drivers and self-driving vehicles. The main problem lies in human error. Because human error is unpredictable, self-driving vehicles need a system by which they can be warned of this error ahead of time.

Both of these reasons speak to the necessity for sophisticated vehicle crash prediction systems. Such a system will give drivers a better understanding of their highly dynamic environment and will give them a better opportunity to react to it. Consequently, I decided to explore crash prediction systems and in this paper, I will elaborate on my two broad approaches: linear approximation (which I frequently refer to as the Naive Crash Prediction algorithm) and ML. The linear approximation approach involves plotting the theoretical trajectories of each of the vehicles in a video and comparing them to see which vehicles have a high potential of colliding with each other. The ML approach involves training a model to predict whether a crash occurs in a given dashcam video by both downscaling the resolution of the videos and extracting numerical information from them as well. Both of these approaches require dashcam video data and I obtained a dataset from the VSLab based in Taiwan for this purpose. The contents of the dataset are described further in the *Dataset* section.

Overall, this study hopes to strengthen vehicle crash anticipation through the analysis of dashcam video footage.

Dataset

For this project, I utilized a dashcam video dataset created by the VSLab based in Taiwan. The videos within this dataset were collected from drivers in the streets of Taiwan and Russia. One of the critical reasons why I chose to use this dataset over others in the research community is the difficulty of its videos. Namely, according to the VSLab, their dataset contains more diverse accidents (motorcycle, car, truck, etc.), crowded streets filled with numerous vehicles and pedestrians, and complicated road scenes (many road signs for example). The strength of these dashcam videos would ensure that the algorithms I developed to predict crashes would be robust.

To elaborate on the technical aspects of the VSLab dataset, each of the dashcam videos was exactly 100 frames in length and was recorded at approximately 25 FPS. This allowed me to generalize my crash prediction algorithm across all of the videos within the dataset. Furthermore, an important quality of this dataset is that for all of the dashcam videos that contain a crash, the crash occurs in the second half of the video. This made it easier to construct test cases for my algorithms.

In total, this dataset contains 620 positive samples (videos that contain a crash) and 1130 negative samples (videos that do not contain a crash) which means that this dataset is slightly unbalanced [1].

	Positive examples	Negative examples	Total
Training set	455	829	1284
Testing set	165	301	466
Total	620	1130	1730

Table 1. Class Distribution of the VSLab Dashcam Dataset

Linear Approximation Approach

In this section I will delve deeper into my linear approximation-based crash prediction algorithm or Naive Crash Predictor.

YOLO Object Detection

YOLO is a state-of-the-art object detection algorithm that is able to identify a variety of different types of objects within a given image. It uses a grid-based searching method paired with a complex deep neural network to both quickly and accurately detect objects within images. In its base form, YOLO has the ability to detect over 80 different

Figure 2. YOLO Object Detection Visualization

types of objects. However, most of them were irrelevant to my study. As a result, I reduced the list of detectable objects down to only traffic-related objects. Specifically, the objects that I set the YOLO algorithm to detect were: person, bicycle, car, motorbike, and bus. Figure 2 shows the YOLO object detection algorithm in action and here only the traffic-related objects are detection (along with a few other auxiliary objects such as traffic lights) [3].

Vehicle Tracking

One of the challenges that I encountered whilst developing my Naive Crash Prediction algorithm was that of vehicle tracking. In essence, I needed a method of uniquely identifying vehicles throughout the entirety of the dashcam video. To do this, I turned to OpenCV's Boosting Multitracker which is able to track multiple objects throughout videos and will give me the identity of the vehicle as well as its position in each frame. I used the YOLO object detection algorithm to identify the vehicles in the first frame of the video and then passed this information along to the multitracker. By providing these initial bounding boxes, the multitracker essentially "locks" onto these vehicles for the rest of the video. Figure 3 shows a visualization created by OpenCV and Figure 4 shows my implementation of the OpenCV Boosting Multi Tracker. An

Figure 3. Multi Object Tracker Visualization

Figure 4. My Implementation of OpenCV Boosting Multi Tracker

important aspect of both of these images is the fact that different objects are surrounded by different colored bounding boxes. This points to the ability of the multitracker to uniquely identify objects throughout the entirety of videos.

For each of the vehicles within the video, I obtained its position at every frame (given to me by the multitracker) and passed this data to my Naive Crash Predictor. One drawback to my approach was that I would not be able to track any new vehicles that entered the video after the first frame. This is because the YOLO object detection algorithm only runs on the first frame and the bounding boxes for all detected vehicles are passed onto the multitracker. Thus, at its current stage, my algorithm can only predict a crash occurring within the vehicles present in the initial frame of a dashcam video.

Trajectory Prediction

From the multitracker, I will have information about the 2D positions of each of the vehicles at each frame for a given dashcam video. To describe the procedure for calculating trajectories more mathematically, we will start with the given initial conditions. Let i be the initial frame and (x_i, y_i) be the position of a particular vehicle in that frame, let $i-1$ be the previous frame with the same information. In order to calculate the position of the vehicle in frame $i+1$, simply find the change in x, $dx = x_i - x_{i-1}$, and the change in y, $dy = y_i - y_{i-1}$. Then add these changes to the position of the vehicle in frame i, so the predicted position in frame $i+1$ is $(x_i + dx, y_i + dy)$. The same principle can be applied in predicting the vehicle's position in the frame $i+k$, which would be $(x_i + k \cdot dx, y_i + k \cdot dy)$.

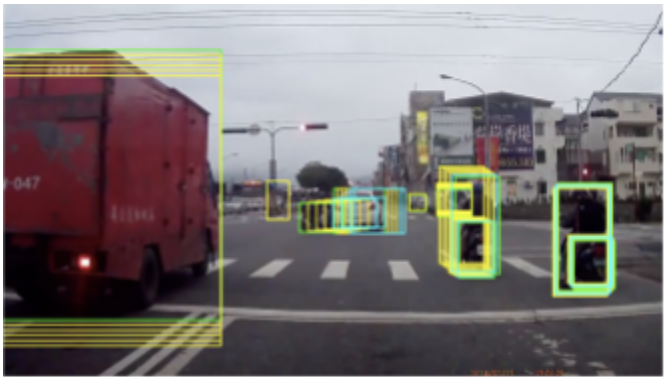

Figure 5. Trajectory Prediction Visualization

To actually use this trajectory prediction algorithm to predict vehicle crashes, we can analyze the positions of vehicles in future frames and determine which are close enough to be deemed "crashing." Specifically, let i be the current frame, and $i + lh$ be the farthest frame into

the future that I will predict vehicle positions for. As a side note, lh is usually 10-12 frames. Then, for all frames k between $i+1$ and $i+lh$, my naive crash prediction algorithm will first forecast the position of all vehicles in frame k and then compare all the predicted positions to determine whether they are below a certain threshold. To do this, given two particular vehicles in frame k, with predicted positions (x_1, y_1) and (x_2, y_2), calculate the distance between them $D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ and check whether it is below a crash threshold value, T_c. I have found that setting T_c = frame.shape[0]/18 (the image height divided by 18) works well. If $D \leq T_c$, mark both vehicles as "crash predicted" with a red bounding box. Figure 5 shows this algorithm in action. Here, the green bounding boxes represent the initial position of the vehicles, yellow bounding boxes represent the predicted positions of the vehicles, and finally, the blue bounding boxes represent the case where the distance between two predicted positions is below a certain threshold. Figure 6, shows the final output of this linear approximation based algorithm, here both the motorcycle and the white sedan are displayed with red bounding boxes indicating that the algorithm predicts that they will crash. However, in Figure 6, one error is apparent. Namely, there is a silver sedan in the background that is incorrectly highlighted with a red bounding box. The reason my algorithm incorrectly thinks that this vehicle will be in a crash is that it can not differentiate the depths of vehicles in the dashcam videos. So even if two cars will pass by each other horizontally without crashing, in 2D space, their trajectories will appear to cross each other and as a result my algorithm will incorrectly predict that they will crash. Thus,

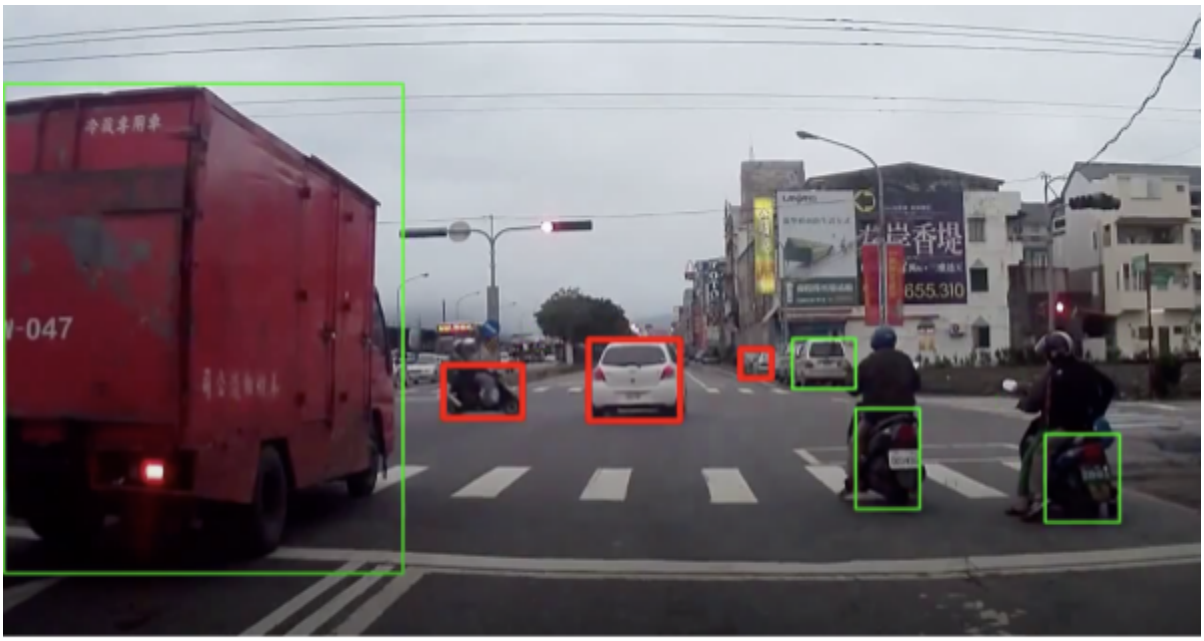

Figure 6. Crash Prediction Without Depth Filtration

the next step I took was creating a method of assigning relative depths to each of the vehicles in a given dashcam video.

Depth Mapping

As I previously mentioned, the main problem with my Naive Crash Predictor was that it could not differentiate the depths of the vehicles in a given dashcam video. As a result, it would incorrectly predict crashes of vehicles that were at drastically different depths, for instance, two vehicles passing each other horizontally.

To resolve this I developed a system of assigning relative depths to vehicles in dashcam videos and prior to labeling two vehicles as "crash predicted" I would ensure that their relative depths are close enough to each other.

The general principle behind my relative depth assignment algorithm was that smaller bounding boxes correspond to vehicles that are deeper in the image. Alternatively, the farther a vehicle is from a dashcam, the smaller it will appear in the image and consequently its bounding box will have a smaller area. My algorithm for calculating the relative depth of vehicles within a dashcam video is as follows, first, choose a reference vehicle at random and determine its bounding box area, A_{ref}. The reference vehicle's relative depth will be set a 1. The relative depth of any subsequent vehicles is simply A_{ref}/A_i, where A_i is the bounding box area of this particular vehicle. Figure 7 displays an example of this algorithm. Above each of the bounding boxes for the vehicles in Figure 7 are their calculated relative depths. Figure 8 explicitly shows how each of the relative depths pictured in Figure 7 was calculated. The white car on the left is the reference vehicle and has a relative depth of 1 and a bounding box area of 9167 pixels squared. The silver car on the very right has a bounding box area of 3536 pixels squared and as a result, its relative depth is

Figure 7. Depth Assignment Visualization

(9167/3536) = 2.59 . The same procedure can be used to determine that the silver car in the middle of the image has a relative depth of 9.26.

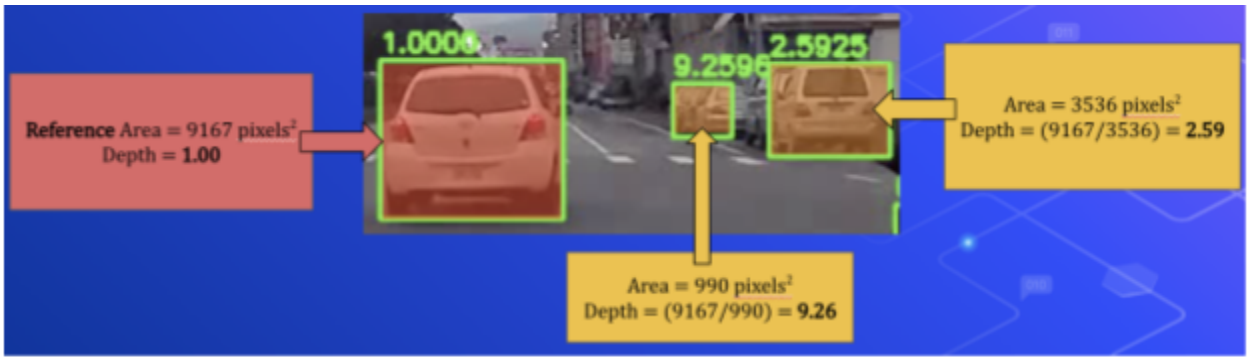

Figure 8. Relative Depth Calculation Example

This depth assignment algorithm, however, does not address the issue of assigning depths between various vehicle types. For instance, motorcycles are smaller than cars which are smaller than trucks, but my algorithm treats these vehicle types as the same. I addressed this by creating depth scaling factors between the various vehicle types. In essence, if a vehicle was a motorcycle, I would multiply its calculated relative depth by a certain scaling number so that its depth would hold more information and meaning. I held cars as my reference vehicle type (scale factor of 1) and through testing I determined the scaling factor for motorcycles to be 0.564 (which means the average motorcycle bounding box area was 0.564 times the average car bounding box area). I created scaling factors for all of the aforementioned vehicle types in the *YOLO Object Detection* subsection. To illustrate this vehicle compliance with an example, if the calculated depth of a motorcycle was 2.4, I would multiply this by 0.564 to obtain the motorcycle's true relative depth which I would compare to other vehicles in the dashcam video.

With respect to using relative depth assignment to filter incorrect predicted crashes, I would determine whether the depths of two vehicles were "close" enough to each other to actually be in a crash. More formally, given the relative depths of two vehicles, d_1 & d_2 , I consider them "close" enough to potentially be in a crash if $\left|\sqrt{d_1} - \sqrt{d_2}\right| < T_d$. Here, T_d is a depth threshold and I have generally found that $T_d = 0.15$ works well in a large number of cases.

Results

Figure 9 illustrates my final Naive Crash Predictor output including both the trajectory prediction and depth filtration. In this image, only the motorcycle and the white car in the center of the image are correctly highlighted with a red bounding box. The best result I obtained using this algorithm was anticipating a crash **12 frames or 0.48 seconds ahead**. The clear limitation of this naive approach is that it approximates a vehicle's trajectory linearly when in fact the motion of vehicles is much more complex. This distinction serves as the basis for my exploration of the ML methods that follow as they will be able to capture the more complex motion of vehicles in dashcam videos.

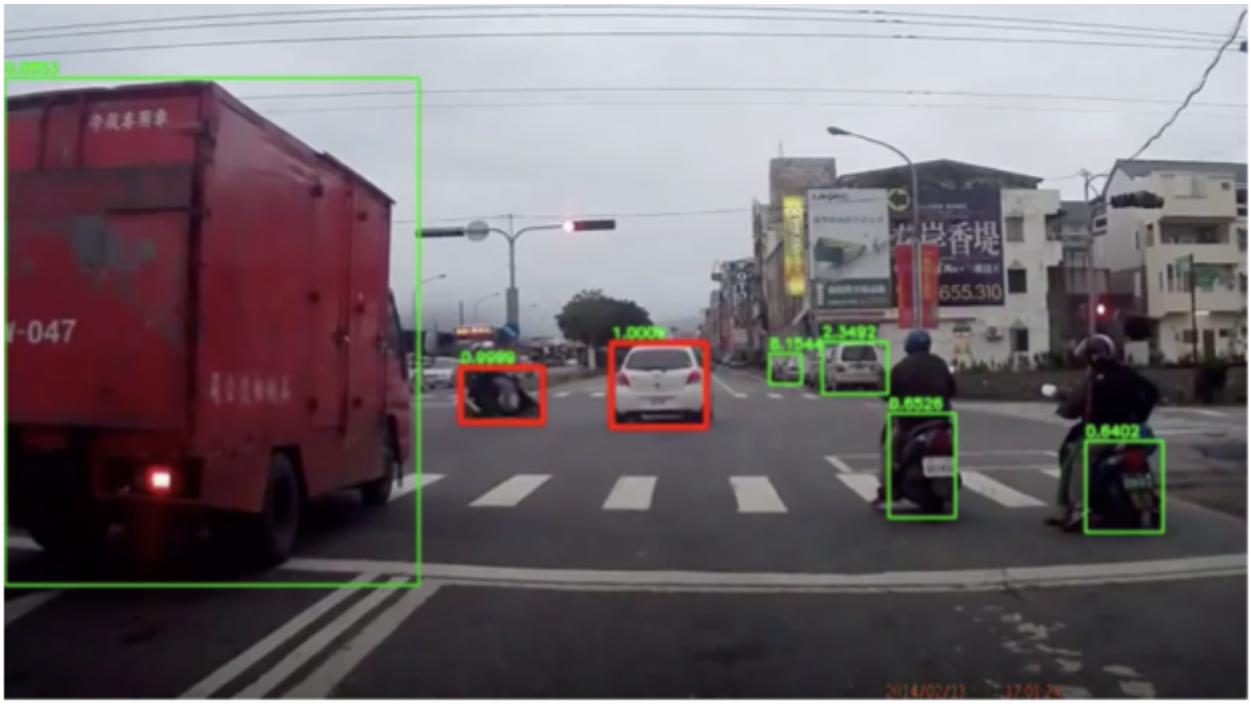

Figure 9. Crash Prediction Example with Depth Filtration

Machine Learning Approach

In this section I will explore the various ML approaches I took in better anticipating crashes in dashcam videos.

Raw Video Input

In this first approach, I used downscaled dashcam videos as my input data. I reduced each of the frames in the dashcam video to a size of (178, 100) pixels for efficiency. Figure 10 shows an example of the appearance of these downscaled frames. I formatted the output data by simply creating a vector of size 100, the number of frames in the dashcam videos within the dataset, and changing the twenty indices leading up to the crash frame to a 1 while keeping the rest 0. So, for instance, if the crash occurred on frame 80, this vector would have zeros from indices 1 to 60 and 81 to 100 and would have ones from indices 61 to 80. This would create a paradigm where the ML model is attempting to predict whether a crash will occur within the next twenty frames in a given dashcam video.

Figure 10. Downscaled Frames as Input Data

I utilized a 2D Convolutional Neural Network with Max Pooling as the basis of my model structure and the best result I obtained was being able to predict a crash **28 frames or 1.12 seconds ahead**. Figure 11 shows the probability vs. frame number output graph for both the Raw Input ML model's prediction (labeled in red) and the ground truth (labeled in blue). In this graph, the output data formatting becomes clearer, all values except the twenty indices leading up to the crash are labeled as zeros and the rest are ones.

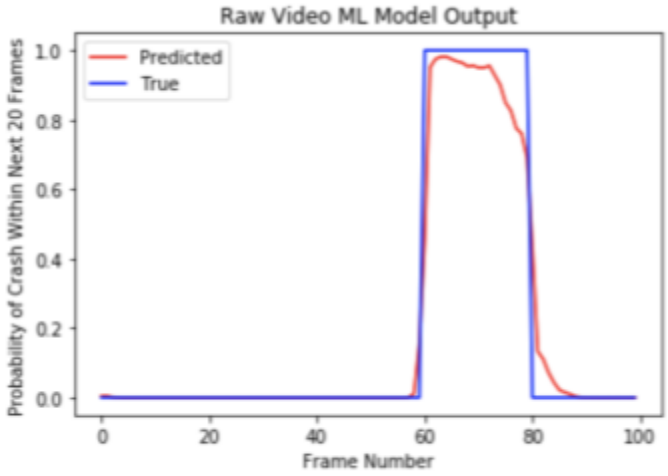

Figure 11. Probability vs. Frame Number for Raw Input ML Model

Discrete Filtered Input

The main problem with the previous ML approach was the sheer size of the data over which the model would have to train. Namely, because I was feeding in raw videos as input, it took several minutes per epoch. On top of this, each of the videos contained random "noisy" information which would only serve to confuse the classifier. As a result, I decided to try another, more filtered approach. Here, instead of the frame itself being the input data, I would instead extract the number of cars, their positions, velocities, and depths from each individual frame and use that as my input data. Figure 12 shows the input data for a single video with 4 detected cars. Each row represents the data for a single frame and the first number is the number of detected vehicles, and the following numbers are the normalized positions, velocities, and relative depths of all four vehicles. The zeros at the end are used for padding purposes to make all video data equal in length. I kept the same output data format as the previous raw ML approach and figure 13 shows the probability vs. frame number for this filtered approach.

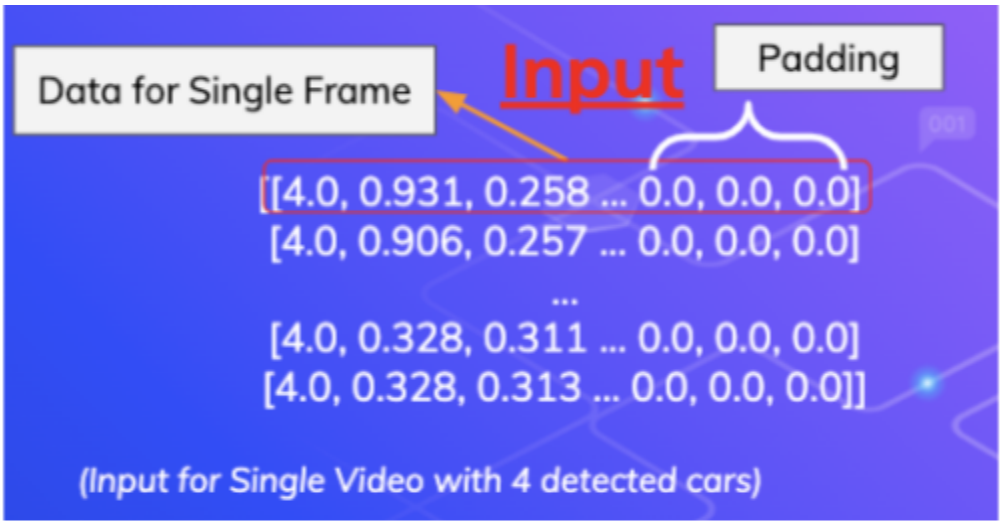

Figure 12. Sample Filtered Input Data

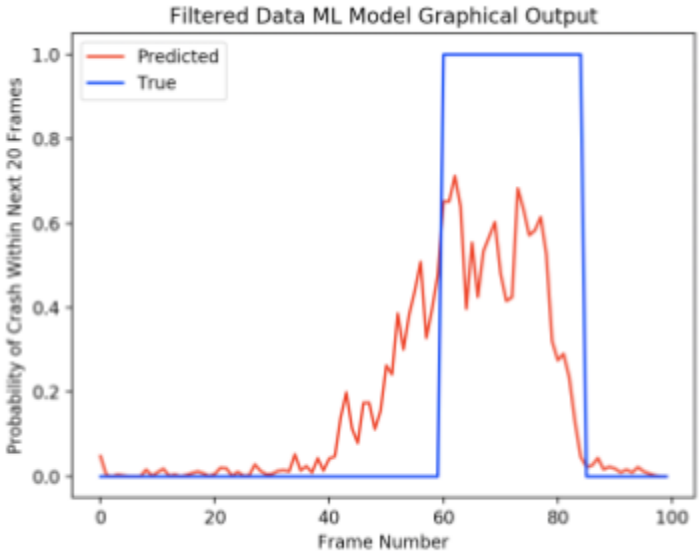

Figure 13. Probability vs. Frame Number for Filtered Input ML Model

For this model, I utilized a 1D Convolutional Neural Network with Max Pooling. Comparing the output graphs for both approaches, it is clear that filtering the videos to discrete numerical values yields a less accurate prediction. And this is further evidenced by the model's performance, its best result was predicting a crash **25 frames or 1 second ahead** while the previous approach was 1.12 seconds ahead. As a result, there is still work that needs to be done in better extracting relevant information from dashcam videos.

Future Works & Conclusion

In summary, this research has given me a lot of insight into how computer algorithms can be utilized to anticipate vehicle crashes in dashcam videos. Although the Naive Crash Prediction approach is reliable, it does not predict crashes far enough ahead to have an impact on a driver's reaction. Thus, improving the sophistication of this algorithm by predicting trajectories more efficiently is essential. With respect to the ML models, I found that the raw approach had higher accuracy and this was expected because using raw dashcam videos as input will give the classifier much more information than simply extracting discrete quantitative features such as the number of cars and their positions. I packaged both the linear approximation and ML vehicle crash prediction algorithms into an easy-to-use GUI (created with Python's Tkinter library) so that future researchers can experiment with them.

Crash Prediction Approach	Best Result
Linear Approximation (Naive Crash Predictor)	12 Frames Ahead (0.48 sec)
ML: Raw Video Input	28 Frames Ahead (1.12 sec)
ML: Discrete Filtered Input	25 Frames Ahead (1.00 sec)

Table 2. Summary of Results

There are many potential avenues other researchers can take to increase the strength of this project. Firstly, creating a more robust system for calculating relative depths. Although using the bounding box areas is a good initial approach, there are many other more mathematically complex ways of doing so which could prove to be beneficial. Additionally, using Recurrent

Neural Networks, which have been shown to have heightened performance on time-series data, in the ML models could potentially improve their performance. Finally, finding a novel method of extracting data from the dash camera videos so that the ML model can still retain accuracy would certainly be advantageous especially in an efficiency sense. Overall, this research has the potential to greatly impact the way we drive. A simple, one-second warning before a crash occurs in all vehicles could drastically reduce the impact of vehicular accidents within society, regardless of whether the driver is a human or robot.

Works Cited

[1] Chan, F.-H. (2017, March 20). Anticipating Accidents in Dashcam Videos. Retrieved November 20, 2019, from https://aliensunmin.github.io/project/dashcam/

[2] Motor Vehicle Deaths Estimated to Have Dropped 2% in 2019. (2020). Retrieved April 20, 2020, from https://www.nsc.org/road-safety/safety-topics/fatality-estimates

[3] Redmon, J., & Farhadi, A. (2018). YOLO: Real-Time Object Detection. Retrieved January 21, 2020, from https://pjreddie.com/darknet/yolo/