Accident Prediction with Dashcam Video: Final Report

Alvin Shek
Electrical and Computer Engineering
ashek@andrew.cmu.edu

Andy Zhang
Electrical and Computer Engineering
andyz@andrew.cmu.edu

Pinxu Ren
Electrical and Computer Engineering
pren@andrew.cmu.edu

Abstract

Object recognition algorithms deals with locating and classifying actions in videos. Since the latest locating algorithms requires less powerful hardware, motivated by the latest state-of-the-art real-time object detector You Only Look Once (YOLO), we aim to test and compare various object detection algorithms. We also re-implement the Anticipating Accidents in Dashcam Videos [7] approach and propose a new modification that applies attention not only to specific objects in an image, but also attention across time on entire images in a video. We demonstrate improved accuracy and comparable runtime during training and inference. Our full presentation is linked here. Our code is linked here.

1 Introduction

Artificial intelligence has made much progress in vehicles, ranging from basic vehicle-assist on freeways to full autonomy. Humans and self-driving vehicles alike need to prioritize their own safe, effective driving behavior. However, how should they handle the unsafe behavior of other vehicles? For our project, we would like to predict accidents a few seconds before they occur and provide some metric of uncertainty on nearby drivers. This will enable drivers to perform emergency, evasive actions just as the accidents happen rather than seconds later. Human drivers especially may not be able to tell that accidents will happen, so our system will assist them.

Object detection is one of the most classic problems in computer vision. The key point is to detect the target's position in the given image, especially to locate and classify the target. Traditional convolutional neural networks (CNNs) have been widely used in target recognition and classification, but they can not locate the target in the image. In addition, classification of images containing multiple targets has always been a major challenge for traditional CNN. The region based CNN (r-cnn) is then introduced into the field of target detection to obtain greater gain. In the region based recommendation framework, r-cnn extracts deep features based on CNN, combines classification and boundary box regression to form a multi task learning model. However, the two-stage target detector has the disadvantages of large amount of calculation, which makes it impractical for real time object detection and classification. The You Only Look Once (YOLO) architecture was later introduced as the first end-to-end object detection network. YOLO is a one-stage detector that only 'look at' the image once. With a CNN it can simultaneously predict multiple boundary boxes and their corresponding class probabilities. It has high accuracy and amazing real-time speed of detection. After YOLO, more algorithms were raised, like Faster RCNN, YOLOv2, YOWO. These algorithms has their advantages and disadvantages.

While Dashcams are popular in many countries such as Korea, Russia, and Taiwan to record accidents and provide evidence during disputes. As a result, many accidents that occur have been recorded with dashcams, making them a promising source of accident data. We will use videos from commercial and non professional dash cameras to track vehicles and forecast collisions. Existing methods have been proposed for our problem focus, so we want to validate their results as well as combine ideas from different approaches into a new, novel architecture. We want to show the effect of object detection method in the accident forecast, and how they could be applied by us to be used in the accident detection tasks.

2 Related Works

2.1 Attention-based Models

The original Anticipating Accidents in Dashcam Videos [7] draws inspiration from [5] by learning to apply attention spatially across a single image. Attention locations are not determined by the prediction model. Rather, an upstream object detector detects vehicles (buses, cars, bikes, motorbikes) as well as people. It crops and feeds each object into a feature extractor. These features inherently represent various boxed regions in the original image, and as the model learns which features are important (order is important when fed in as a matrix), we can visualize this attention on each bounding box.

Attention is simply a normalized scalar weight applied to each object in the scene, and the sum of attention values should sum to one. Suppose, for a given video frame at time t in a video, there are J objects, each represented by a D-dimension feature vector: $a_t^j, j <= 19$. The 19 is a max number of objects the model can handle since we feed in fixed matrix inputs of shape (B x 100 x 20 x D) for B batch size. The 20th entry is a feature representing the entire image, not any particular object.

This paper uses LSTM's, or Long Short Term Memory, to capture historical information. An LSTM is fed in the current input x_t as well as its previous hidden state, h_{t-1} to produce the a new hidden state h_t. At the same time, h_{t-1} is used to generate attention weights for the objects in the current frame's image:

$$e_t^j = g(W_h h_{t-1} + W_x x_t^j + b)$$
$$\alpha_t^j = \frac{exp(e_t^j)}{\sum_j exp(e_t^j)}$$

In the first step, $g(z)$ is the nonlinear activation function, tanh. In the second step, the logits e_t^j are normalized with a softmax function to produce final attention weights α_t^j. This is so they all sum to 1. One key point to note is that at any given time frame, there often will not be 19 objects present. In this case, the values of the empty spots in the input matrix are masked with zeros. This makes sure they do not get any attention. Intuitively, one can think of $\alpha_t^j = p(a_t^j | h_{t-1})$ as the probability that the jth object is important in accident prediction. h_{t-1} fulfills the Markov Property and captures all previous historical information.

Next, all the object features are weighed and summed together, hence the term "soft attention":

$$\hat{a_t} = \sum_{j=1}^J \alpha_t^j a_t^j$$

This is then concatenated with the full image feature, the 20th entry of the original input matrix:

$$X_t = [a_t^{20}; \hat{a_t}]$$

This final feature vector is transformed with h_{t-1} in a nonlinear affine transform (Linear layer followed by softmax activation) to output two values representing $p(accident)$ and $1 - p(accident)$ respectively. In practical usage, it is a hyperparameter that a driver would tune to set the threshold for what is noise and what constitutes a real risk.

2.2 CNN related two stage detector

R-CNN [9] is one of the most important and groundbreaking works to introduce CNN into the field of object detection. It uses Selective Search to generate potential bounding boxes in images, and then classifies them. Then it introduces **Fast R-CNN** [10] and **Faster R-CNN** [15], by sharing computation and using neural network to propose regions instead of selective search to reduce computing time. These three algorithms belong to two-stage target detector that have the region of interest generated by the region recommendation network, then the target is classified by pipeline, and the boundary box regression is performed. The algorithm has high precision, but it is often slow in practical application.

2.3 YOLO detectors

The **YOLO**[11] architecture eliminates the need for two-stage detector and it is the first end-to-end object detector. YOLO regards object detection as a simple regression problem by acquiring an input image and learning its class probability and boundary box coordinates. It's incredibly fast and produces competitive results in real time. However, because YOLO processes global images, it is difficult for YOLO to accurately locate small objects, such as a flock of birds. Although this might not be a common situation in the context we want to research about.

Based on YOLO box, the recall rate for **YOLOv2**[12] is improved because it uses anchor boxes to predict bounding boxes. In order to improve the recognition accuracy of small targets, a passthrough layer was used to extract fine-grain characteristics. YOLOv2 also considers multi-scale training to enhance the robustness of the model to images of different scales. In order to facilitate the speed of object detection, YOLOv2 compressed the feature representation into Darknet-19 network using global average pool and use a small sized filter to reduce floating-point arithmetic.

3 Contribution

3.1 Temporal-based Attention for Accident Prediction

Our first step in analyzing Anticipating Accidents in Dashcam Videos [7] was to simply visualize the output of the model. We were able to see specific objects given higher attention as shown in the below figure 1:

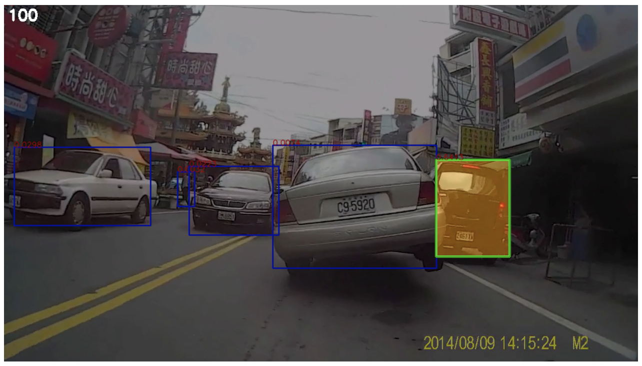


Figure 1: Object with green bounding box given higher attention during accident.

We noticed in many other scenarios the model would just wrongly place attention on vehicles that were completely still. Then the model would predict high probability of accident when no accident was present because it kept hallucinating that the still object was in an accident. This is shown below:

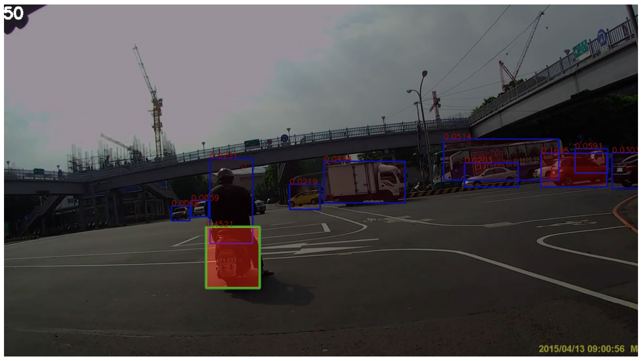


Figure 2: Model hallucinating that still object in front is in an accident.

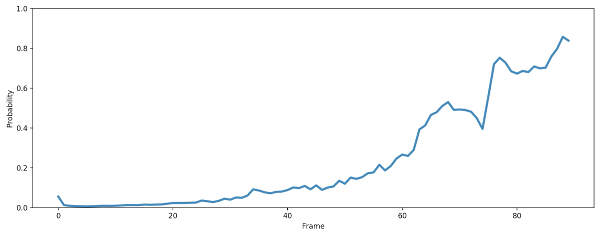


Figure 3: Due to constant hallucinations, model predicts high probability of accident.

We decided to introduce a new modification: learn attention across time on the full image features a_t^{20}. Our proposed model overall still retains the original paper's spatial object attention, but now additionally learns which of the previous k image frames are most important for predicting an accident in the current frame. This idea drew inspiration from the Natural Language Processing domain and the end-to-end Speech-to-text model of Listen, Attend, and Spell [8]. Such models learn to gradually shift their focus across time in an audio recording to correctly generate text.

We first create an Encoder model that takes as input a window of the past K video frames including the current frame: $[a_{t-k}^{20}, a_{t-k+1}^{20}, ...a_t^{20}]$. Note that these are the full image frame features, not specific object features. Next, this entire sequence is passed into Pytorch's LSTM module, which internally handles the use of hidden states for us. The LSTM makes an output at each timestep, and in total, we get a new output matrix of shape (B x K x H) where H is the hidden dimension size and K is the length of our time window. This entire matrix is then fed into two separate linear layers to generate a key W_K and value W_V matrix respectively, each of size (B x K x H').

At any given timestep, the original paper [7] would simply use the current full image feature: a_t^{20} and concatenate that with the combined object features. Instead, we multiply a_t^{20} with key matrix W_K followed by a softmax to get normalized attention weights W_e. In [8], the full image feature would be called a "query". The learned key matrix in essence is multiplied with the current image features to determine which of the past K frames are most important. The output weights are then batch-multiplied (bmm) with the value matrix, which essentially performs a weighted sum of the past K frames' information:

$$\hat{a}_t^{20} = bmm(W_e, W_V)$$

This can be summarized in the diagram:

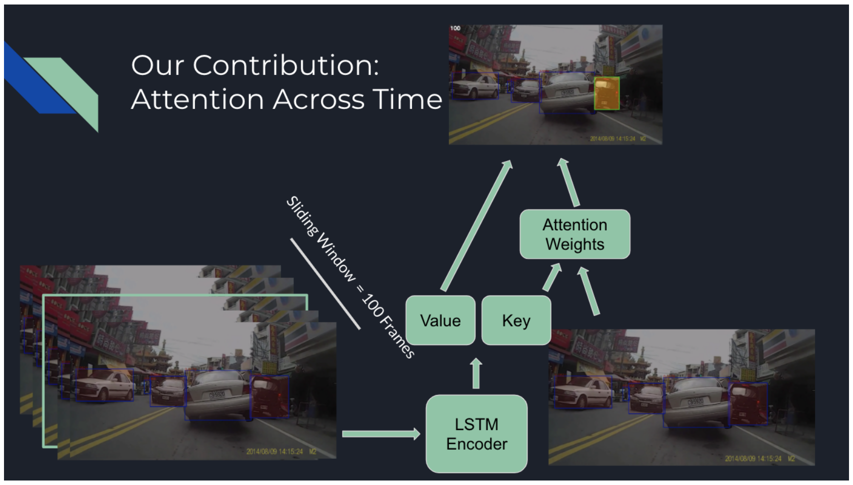


Figure 4: The above diagram shows how attention across images in time is computed.

Below is the loss function used, which is time-weighted CrossEntropyLoss:

$$J = \sum_t^T [-ye^{t-k}log(p) - ylog(1-p)]$$

where y = 0 (no accident) or 1 (accident) represents a sequence of T video frames before an accident. k denotes the timestep or frame index where the accident occurs in the entire sequence. t is a given frame's timestep or index. p is the model's predicted probability of an accident occurring at that timestep.
The above is basically a modified version of Cross-entropy loss, where only the positive examples with accidents have the additional weights. As t approaches k, the exponent will approach 0, and the entire weight will approach 1. If t « k, then the entire weight will approach 0. This intuitively means we should penalize failures to predict accident the closer we get to the accident. If we are far from the accident, we shouldn't be penalized that much for failing to predict. For a given sequence of T video frames, all frames in that video are assigned the same label (accident or not).

Below summarizes the network sizes.

Table 1: Network Parameters

n-input	n-detection	n-hidden	n-img-hidden	n-att-hidden	n-classes	n-frames
4096	20	512	256	256	2	100

n-input denotes D, the input feature dimension.
n-detection = 1 + J, where J is the max number of objects tracked in a given frame.
n-hidden is the hidden size of the final LSTM that takes in the concatenated full image and object features: $X_t = [\hat{a}_t^{20}; \hat{a}_t]$
n-img-hidden is the output size of the features in the Key and Value linear layers.
n-att-hidden is the transformed feature size of the object features.
Notice that n-hidden = n-img-hidden + n-att-hidden since the full image and object features are concatenated.
n-classes = 2 since we predict $p(accident)$ and $1 - p(accident)$.
n-frames = K = 100, our time window length.

Below summarizes the hyperparameters we used. We stuck with the paper's own original parameters:

Table 2: Training Parameters

learning-rate	n-epochs	batch-size	display-step
0.0001	20	10	10

Table 3: Final Performance

Model	Average Precision	Prediction Time Before Accident	Runtime Per Epoch
Baseline	0.64	1.149s	498.7s
Our Method	0.667	1.257s	583.4s

As you can see above, our model has higher average precision and predicts accidents on average earlier. However, our model needs to compute new attention values across the past K video frames, and thus takes an extra 80 seconds approximately per epoch. Given a batch size of 10 videos and 128 batches in total, processing a video of 100 frames takes approximately 0.0625 seconds longer with our model.

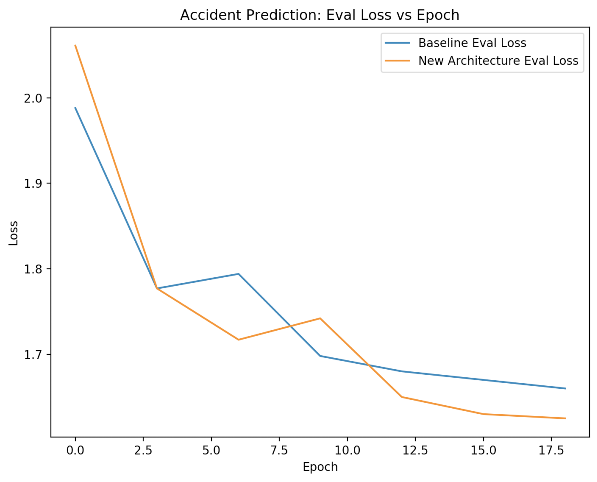


Figure 5: Evaluation Loss vs Epoch. At the 18th epoch, we achieved an evaluation loss of 1.625 compared to the baseline model's loss of 1.66

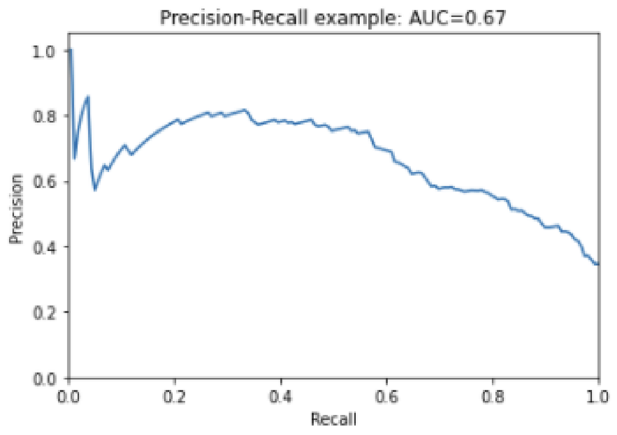


Figure 6: Precision vs Recall. One can interpret this by picking a desired level of recall (example 0.8) and seeing the corresponding precision.

4 Experiments

We also re-implemented other object detection method such as YOLOv1. The core idea of YOLO series of detection is to use the entire picture as the input of the neural network, and directly return the position of the bounding box and the category to which the bounding box belongs in the output layer, which facilitate the detection process. This method's latest version, YOLOv4, is the state-of-the-art method for object detection. We want to have a better understanding of this method through implementing this method. We use the ResNet50 as the backbone of YOLO and trained it on the dataset of voc2007 and voc2012. We have a reasonable result of accuracy, which is about 60 percent. The training and validation loss of the first 8 epochs are shown in Fig.7. Using Darknet-19, a convolutional neural network framework, we also trained on the voc2007 and voc2012 training sets using YOLOv2. The average loss and mAP during training are shown in Figure 8. Testing the trained model on voc2007's testing set, we obtained a mean average precision of 68.38% at an IoU threshold of 50%.

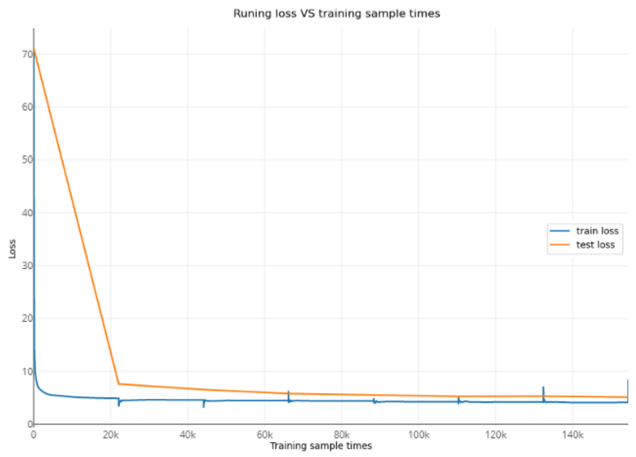


Figure 7: Training and testing loss vs training batches.

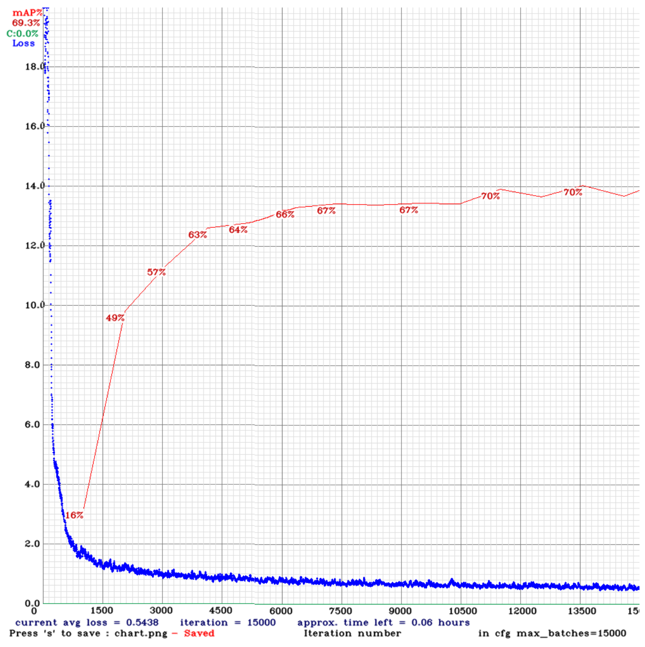


Figure 8: Training loss vs training batches for YOLOv2 on VOC 2007 Test Set

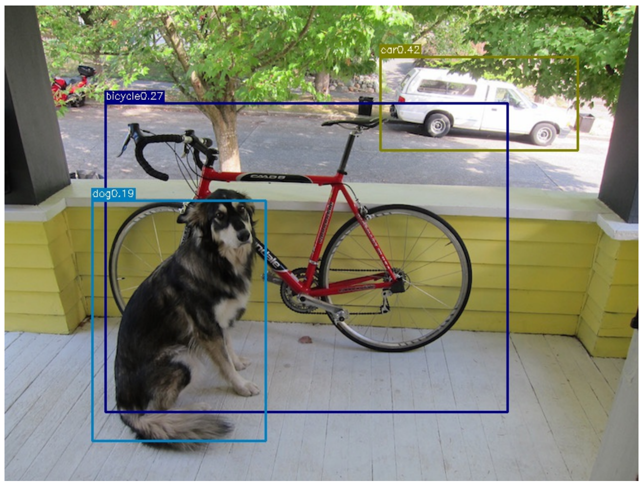


Figure 9: One result output from YOLOv1.

Recently in April 2020, YOLOv4 was introduced by Alexey Bochkovskiy with new features that was shown to achieve 65.7% AP50 on the MS Coco dataset with a speed of 65 FPS on a Tesla V100. [13] As our goal was to achieve as near real-time object detection as possible without sacrificing too much precision, we decided to train and test YOLOv4 on our dataset. As a one-stage detection method, YOLO would ideally reach a minimum of roughly 25-30 frames per second during model evaluation and be able to follow, evaluate, and detect objects during a dashcam's operation during vehicular operation. On the other hand, we also wanted to look at a two-stage detection method as well to compare YOLOs speed and accuracy. After reading about Facebook's Detectron2 [14] object detection implementation and inclusion of popular object detection models such as Fast R-CNN, Faster R-CNN, and RetinaNet, we decided to train and test on our data set using detectron2's faster R-CNN model based on the ResNet-50 model. Detectron claimed that their R50 Faster R-CNN model had a training time of 0.209 seconds per iteration, 0.038 seconds per image for each inference, and a box AP of 40.2. [14] This way, we could have a valid comparison between one of the fastest two-stage detection models and a fast but relatively accurate one-stage detection model.

YOLOv2 was the initial improvement for the original YOLO method, and YOLOv4 was built with Darknet53 as its backbone. [13] Darknet53, a convolutional neural network 53 layers deep that was first introduced with YOLOv3, was ideally faster than Resnet and used residual connections that have already been trained on a multitude of images from ImageNet. To attempt to introduce some variation in training and detection, we changed the resolution size for YOLOv4's model. By default, the network takes in images and converts to a square-sized resolution (e.g. 416 by 416 or 608 by 608) that is a multiple of 32. Lower resolutions should lead to faster training speed and evaluation/inference times with decreased precision while higher resolutions would lead to slightly slower training speeds and evaluation/inference times with better precision for smaller objects as nothing is shrunk down. Finally, we also tried training with a YOLOv4-Tiny model that was the YOLOv4 model but with only 2 YOLO layers instead of 3 on top of the Darknet architecture. We hoped to reduce one entire layer of output and drastically improve speed but were not exactly sure of the definite tradeoffs for speed/precision that would occur.

Our dataset, which was the Predicting Dashcam Accidents' dataset, consisted of 620 videos that showed short dashcam recordings around Taiwan with a portion being recorded accidents. [1] We split the data set by turning each video into each of its 100 separate contained frames, then randomly selecting 10,000 images for training and 2000 each for testing and validation. We settled on 10,000 images due to our limited time and GPU compute ability which meant that we had to have a reasonably-sized data set that we could train on in time to procure adequate results. More importantly, we also had to be sure to select random images because as we split the videos up frame-by-frame, a lot of images would be extremely similar as they were mere milliseconds apart, which could lead to potential over-fitting. To combat this, we selected images for each section (train, test, validation) entirely separate from the others (e.g. an training image part of video 55 would mean video's 55 other frame images would not be used in test/valid). However, due to this constraint, our data set ultimately became a little small due to the need for frame/image variance and as a result some slight over-fitting may have occurred to only training on 10,000 images. In the future, a more complete result from these object detection methods could be potentially obtained on a similar data set with many more images that are not so closely related.

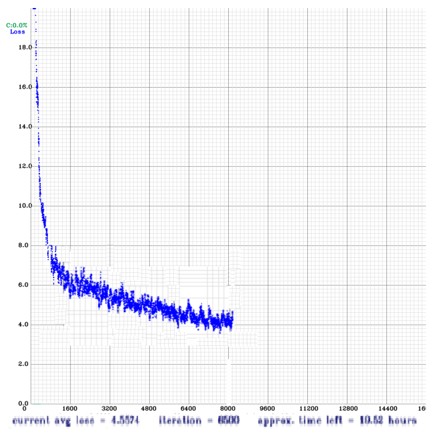


Figure 10: Loss Graph for YOLOv4 (416 Resolution) over 6500 iterations or (6500 / (10000 training / 64 batch size)) ≈ 41.6 epochs

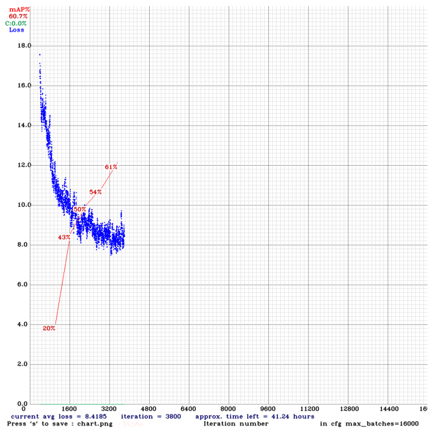


Figure 11: Loss Graph for YOLOv4 (608 Resolution) over 3800 iterations or (3800 / (10000 training / 64 batch size)) ≈ 24.32 epochs

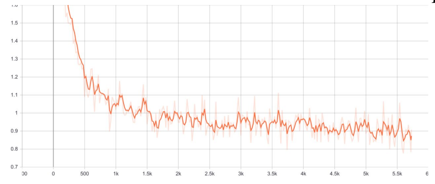


Figure 12: Loss Graph for Faster R-CNN over 6000 iterations or 10000 trainig / 64 batch size = 64 epochs

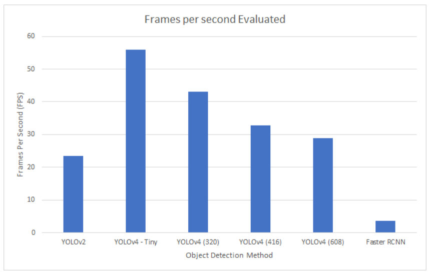


Figure 13: Frames per Second vs. Object Detection Method/Variation

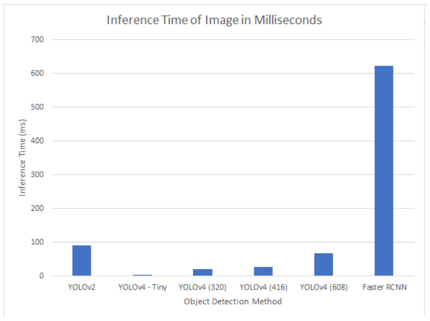


Figure 14: Inference Time (ms) vs. Object Detection Method/Variation

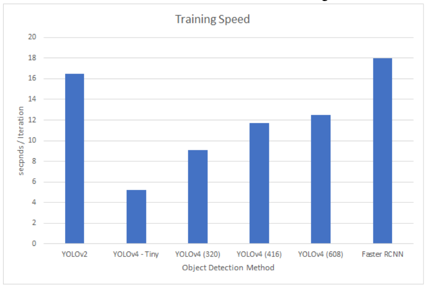
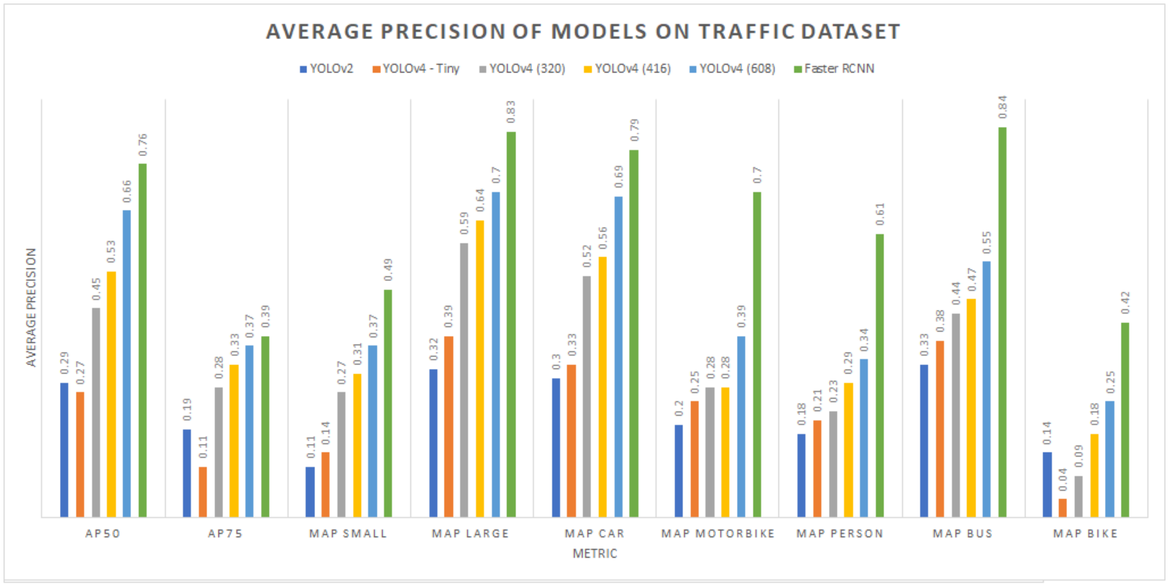


Figure 15: Training Speed(s/iteration) vs. Object Detection Method/Variation

During our training, our loss graphs appeared to follow the correct decline. Unfortunately, due to our limited GPU (RTX 2080) and time, we had to end training early for several of the YOLO model variants before they fully converged.

From our training results, we can see that YOLO clearly has a higher FPS than Faster-RCNN when tested on traffic videos. (Figure 13) In fact, YOLOv4 at 416 Resolution (32.78 FPS) is roughly 900% faster than Faster R-CNN (3.6 FPS) and almost 2000% faster for inference time (28 ms vs. 622 ms) on average. (Figures 13 & 14) Clearly, if we are aiming for near real-time detection, Faster R-CNN is not fast enough to make the cut. Looking a bit more closely at the YOLO variants, we see that YOLOv2 lags behind in training speed, inference time, and FPS while for YOLOv4, as you increase the resolution, the model has faster Inference Time but lower FPS and Training Speed.

As seen in Figure 16, our precision results are equally as telling. The Faster R-CNN had the highest mean average precision for nearly every category, but as mentioned previously, was incapable of being run in real-time, rendering it useless for live dash-cam predictions. YOLOv2 predictably was worse than most of the YOLOv4 variations, but was out performed YOLOv4-Tiny on some metrics. For YOLOv4, we see that as the network resolution increased, the mean average precision rose along with it. At 608x608, we obtain a reasonable mAP of 0.66, while a mAP of 0.45 is obtained at 320x320. Interestingly, YOLO seemed to do relatively well for the large objects and specifically the buses and cars. However, all the YOLO versions suffered a drastic drop in mAP when attempting to identify smaller objects like bikes, motorbikes, and people. Notably, even at 608x608 resolution, YOLOv4 only had a mAP of 0.25 for bikes and 0.37 for all small objects in general. On the other hand, the Faster R-CNN model performed relatively well on smaller objects with smaller objects but suffered a big dropoff in comparison to large objects. Another interesting fact was when the threshold was set to 75%, even the Faster R-CNN model dropped to 0.39 and only 0.02 more than YOLOv4 at 608x608. It appears that both models have trouble detecting objects with certainty and smaller objects especially. Finally, the previously mentioned fastest model, the YOLOv4-Tiny variation with only 2 YOLO layers, did not fair well in any of the mAP measurements with a 0.29 mAP overall at 50% threshold. These trends are further confirmed by looking at the average precision vs. FPS graph, where we see the Faster R-CNN model is clearly the most accurate at mAP@50 but extremely slow. Else, the YOLO models are extremely fast and close to real-time but the lowering the resolution or number of layers caused a big dropoff in mAP.



Metric	YOLOv2	YOLOv4 - Tiny	YOLOv4 (320)	YOLOv4 (416)	YOLOv4 (608)	Faster RCNN
AP50	0.29	0.27	0.45	0.53	0.66	0.76
AP75	0.19	0.11	0.28	0.33	0.37	0.39
MAP SMALL	0.11	0.14	0.27	0.31	0.37	0.49
MAP LARGE	0.32	0.39	0.59	0.64	0.7	0.83
MAP CAR	0.3	0.33	0.52	0.56	0.69	0.79
MAP MOTORBIKE	0.2	0.25	0.28	0.28	0.39	0.7
MAP PERSON	0.18	0.21	0.23	0.29	0.34	0.61
MAP BUS	0.33	0.38	0.44	0.47	0.55	0.84
MAP BIKE	0.14	0.04	0.09	0.18	0.25	0.42

Figure 16: Mean Average Precision at 50% and 75% thresholds and also for small/large objects and for each specific class in our data set

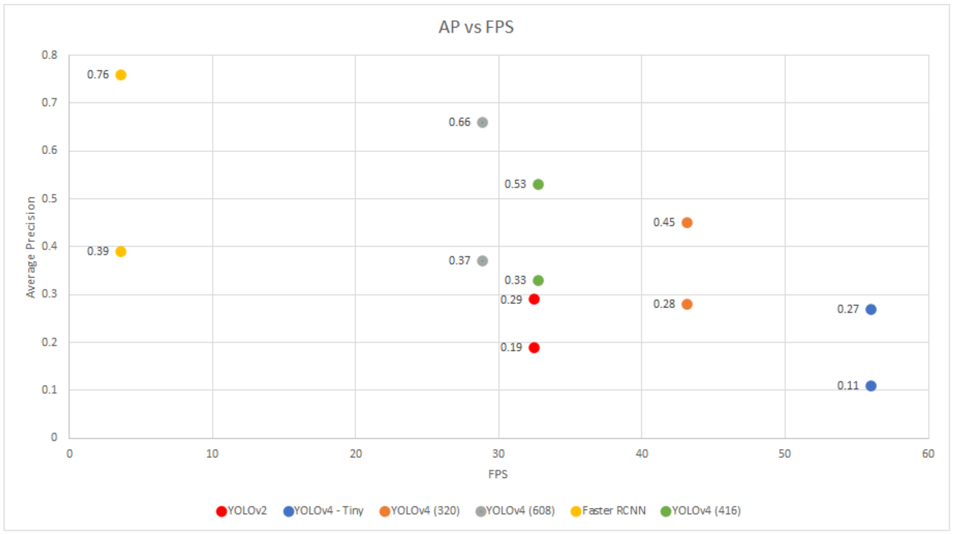
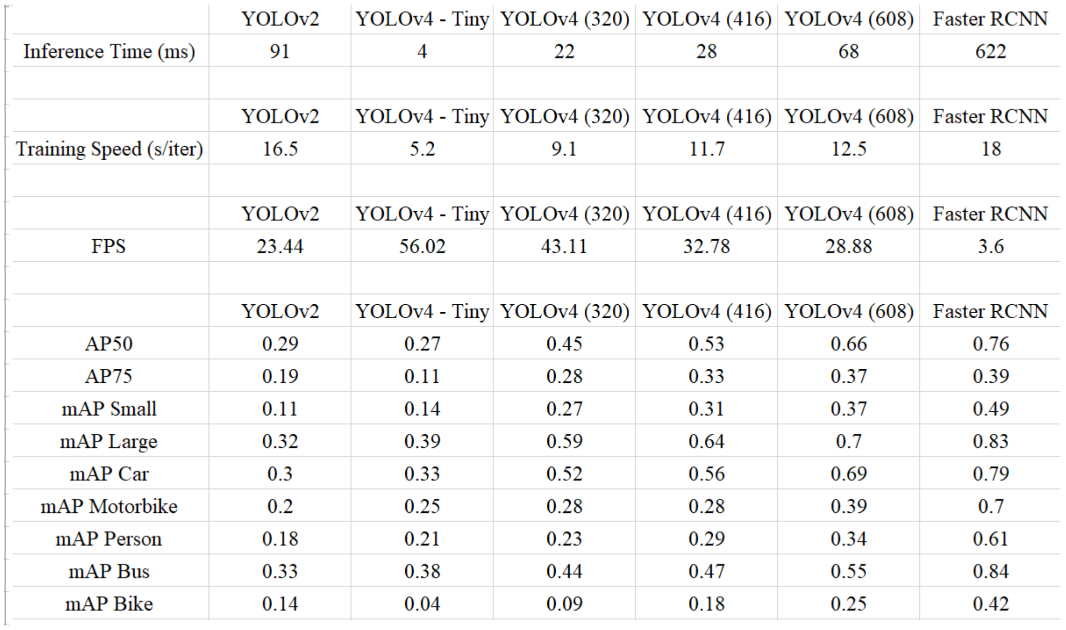


Figure 17: Average Precision vs. FPS

5 Datasets

Dashcam Accident Dataset is the dataset we use. The dashcam is a cheap camera that can be installed in the car to record the visual observation of the street level from the driver's point of view. In Russia, Taiwan and other places, almost all new cars have been equipped with dashcam in the past three years. As a result, a large number of dashcam videos are shared on video sharing sites such as YouTube. The research team in [7] collect dashcam videos shared online from many users, targeting accident videos with annotation of address information or GPS location. In this way, they collected videos that all have high-quality (720p resolution). The dataset includes 678 videos taken in six major cities in Taiwan, which include: 42.6% of motorcycles hit cars, 19.7% of cars hit cars, 15.6% of motorcycles hit motorcycles, and 20% of other types. Almost all the big cities on the west coast of Taiwan are covered. Then the team in [7] did hand annotation with temporal locations of accidents and



	YOLOv2	YOLOv4 - Tiny	YOLOv4 (320)	YOLOv4 (416)	YOLOv4 (608)	Faster RCNN
Inference Time (ms)	91	4	22	28	68	622
	YOLOv2	YOLOv4 - Tiny	YOLOv4 (320)	YOLOv4 (416)	YOLOv4 (608)	Faster RCNN
Training Speed (s/iter)	16.5	5.2	9.1	11.7	12.5	18
	YOLOv2	YOLOv4 - Tiny	YOLOv4 (320)	YOLOv4 (416)	YOLOv4 (608)	Faster RCNN
FPS	23.44	56.02	43.11	32.78	28.88	3.6
	YOLOv2	YOLOv4 - Tiny	YOLOv4 (320)	YOLOv4 (416)	YOLOv4 (608)	Faster RCNN
AP50	0.29	0.27	0.45	0.53	0.66	0.76
AP75	0.19	0.11	0.28	0.33	0.37	0.39
mAP Small	0.11	0.14	0.27	0.31	0.37	0.49
mAP Large	0.32	0.39	0.59	0.64	0.7	0.83
mAP Car	0.3	0.33	0.52	0.56	0.69	0.79
mAP Motorbike	0.2	0.25	0.28	0.28	0.39	0.7
mAP Person	0.18	0.21	0.23	0.29	0.34	0.61
mAP Bus	0.33	0.38	0.44	0.47	0.55	0.84
mAP Bike	0.14	0.04	0.09	0.18	0.25	0.42

Figure 18: Precision and Speed Metrics

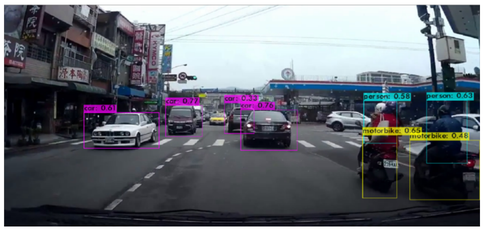


Figure 19: YOLOv4 Prediction on Test Image

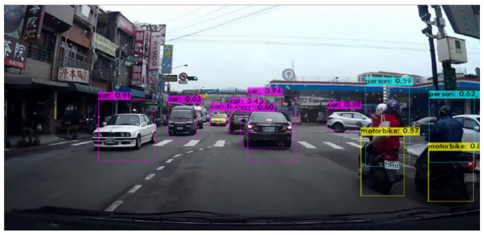


Figure 20: Faster R-CNN Prediction on Test Image

the moving objects in each video. The below table lists the distribution of different objects in the scene:

Table 4: Dataset constitution parts

Car	Motorbike	Person	Bus	Bike	**Total**
20829	15091	13436	1604	185	51145

6 Discussion and Future works

After the midterm progress check, we have finally fixed the issues with our own implementation of the paper's architecture. We found two issues:
1. Videos are each 100 frames in length with the accident (if any) always being the last frame. In terms of a classification problem, all frames of a video containing an accident will have a label of 1. Frames of a video without an accident will have a label of 0. Pytorch's CrossEntropyLoss() function takes in predictions of shape (N x C) where N is the number of samples and C is the number of classes. C = 2 in our case. Since we have one label per video, we need to broadcast this from shape (1 x 1) to (100 x 1) so all frames use this label target. We did not reshape the labels correctly. We wanted this format: [video1-frame1, video1-frame2, ... video1-frame100, video2-frame1, ...] but instead had this format: [video1-frame1, video2-frame1, ... videok-frame1, video1-frame2, video2-frame2...].

2. Another core issue was that we were not masking attention correctly. Each frame, we have a maximum of 20 objects tracked at any given moment and their extracted features. So our features are of shape 20 x D. However, there can be less than 20 objects actually present in a frame, so only k <= 20 features will be valid. When generating attention values for each feature, we apply Softmax() function so attention weights over all objects sum up to 1. This is similar to following the total probability law that the probability of all objects being useful should sum to 1. The bug was that we were computing first softmax, and after masking out the non-existent object attention weights. This means attention weights do not all sum to 1, and the non-zero, remaining attention weights are wrong.

We noticed that the loss would plateau and converge around only 20 epochs. We believe that quick convergence, but only marginal better performance can be attributed to our relatively small layer sizes. Specifically, our rich input features have dimension (4096 x 1) and contain both visual and motion features. We transform this down to only (256 x 1) features, and this sharp reduction may be a bottleneck that limits how much information our model can obtain from the data.

Another high-level concern we had was that the raw video feed of a dashcam video can be very noisy from dirt on the lens or motion blur from the car shaking. One possible future step can be to smooth out the original input video feed and perform other data cleansing techniques. In this project, we were already provided the extracted features. Another possible work is to compare the features of other classification models. This original paper uses VGG-16's backbone to extract features from images. It would be interesting to compare performance using other models' extracted features, for example ResNet50 or MobileNetv2.

7 Division of work

Alvin Shek: draft reports, review literature and present. Develop the time-based attention for accident prediction and re-implement original paper architecture in Pytorch from Tensorflow. Write scripts to convert dataset into COCO format.

Andy Zhang: draft reports, review literature and present. Run YOLOv4

Pinxu Ren: draft reports, review literature and present. Run YOLOv3 baseline models.

References

[1] Bao, Wentao, et al. “Uncertainty-Based Traffic Accident Anticipation with Spatio-Temporal Relational Learning.” ArXiv.org, 1 Aug. 2020, arxiv.org/abs/2008.00334.

[2] Maqueda, Ana I., et al. “Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars.” ArXiv.org, 4 Apr. 2018, arxiv.org/abs/1804.01310.

[3] Fatima, Mishal, et al. “Global Feature Aggregation for Accident Anticipation.” ArXiv.org, 16 June 2020, arxiv.org/abs/2006.08942.

[4] Suzuki, Tomoyuki, et al. “Anticipating Traffic Accidents with Adaptive Loss and Large-Scale Incident DB.” ArXiv.org, 8 Apr. 2018, arxiv.org/abs/1804.02675.

[5] Xu, K., et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." ArXiv, 2015, https://arxiv.org/pdf/1502.03044.pdf

[6] Jain, A., Koppula, H.S., Raghavan, B., Soh, S., Saxena, A.: Car that knows before you do: Anticipating maneuvers via learning temporal driving models. In: ICCV. (2015) https://arxiv.org/abs/1504.02789

[7] Chan FH., et al. "Anticipating Accidents in Dashcam Videos." ACCV 2016. Lecture Notes in Computer Science, vol 10114. Springer, Cham. https://doi.org/10.1007/978-3-319-54190-7_9

[8] Chan, W., et al. "Listen, Attend and Spell" arXiv 2015, https://arxiv.org/abs/1508.01211

[9] Girshick, R., et al. "Rich feature hierarchies for accurate object detection and semantic segmentation" arXiv 2013, https://arxiv.org/abs/1311.2524

[10] Girshick, R. "Fast R-CNN" arXiv 2015, https://arxiv.org/abs/1504.08083

[11] Redmon, J. "You Only Look Once: Unified, Real-Time Object Detection" arXiv 2016, https://arxiv.org/abs/1506.02640

[12] Redmon, J "YOLO9000: Better, Faster, Stronger" arXiv 2016, https://arxiv.org/abs/1612.08242

[13]A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020; http://arxiv.org/abs/2004.10934.

[14] Y. Wu, A. Kirillov, F. Massa, R. Girshick, and W. Lo, “Detectron2” 2020; https://github.com/facebookresearch/detectron2.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-timeobject detection with region proposal networks. InAdvances in neural information processingsystems, pages 91–99, 2015.