Accident Detection On the Edge

Saifullah Rais, Matthew Potts, Kyle MacIlvain

Date:April 16, 2019

We propose an accident detection algorithm which can be deployed on edge devices like smart surveillance cameras. The solution is based on transfer learning, using various pre-trained image classification algorithms to detect accidents. The training is supervised and requires sourcing of surveillance footage and labelling of video frames. With a dataset of 20,000 video frames, the best model exhibits 96% accuracy on a held-out set. In the wild, we saw high precision (80%) and relatively low recall (60%) with an f1-score of 0.7. With an inference time of 0.7 sec/frame (Inception V3), the fastest frame rate currently possible is 1-2 fps. On the Jetson TX2, inference is registered at 25-30 fps using the Jetson Deep Learning SDK.

Given the context, further work is needed in improving inference time and recall rates. Collisions occur within fractions of a second. This implies the possibility of missing an accident due to slower inference time. Lower recall rates, on the other hand, implies that accidents may go undetected. Ideally, we would prefer to err on the side of caution. While pruning and smaller networks could improve inference time, we believe an object detection approach could improve recall rates.

INTRODUCTION

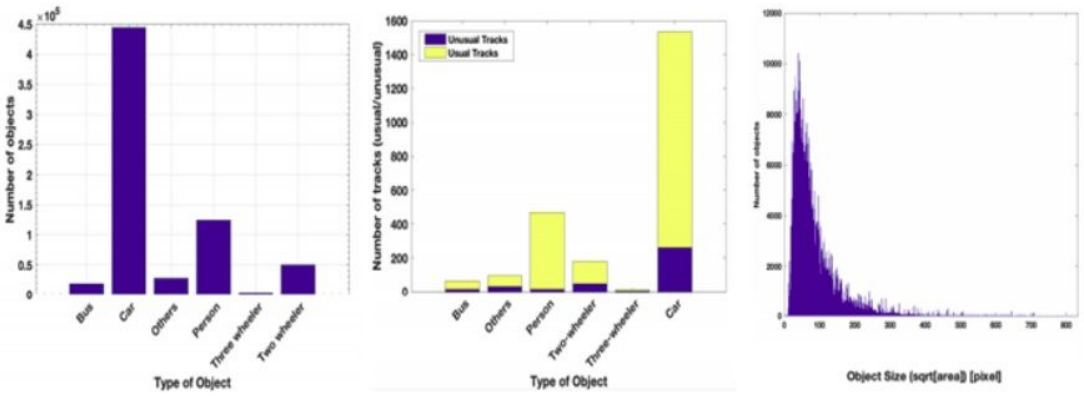
Nearly 1.3 million people are killed globally in car accidents on a yearly basis and it's estimated that accidents will be the 5th leading cause of death by 2030.[1] Cases of hit-and-run, which have higher fatality rates, are on the rise too. In India, almost 15%[2] of fatalities were associated with hit-and-run incidents. In US, hit-and-run deaths are at an all-time high.[3] Given that survival rate amongst severe car crashes can be highly dependent on the length of time between when an accident occurs and when victims arrive at the hospital, immediate alerting of a car crash to local emergency services would aide to minimize this timespan and improve chances of survival and reduce the risk of a resulting disability. As cars become smarter and better connected in the future, such notifications could also be sent directly to nearby cars to reroute traffic thus clearing the way for emergency personal and further improving response time.

While the US has over 50 mn surveillance cameras, some major global metropolitan cities like Mumbai have more than 4,700 surveillance cameras installed. While these cameras are being used retrospectively due to the real-time monitoring bottleneck, deep learning solutions at the edge could save the day.

We propose a deep learning on the edge solution in which a video classification algorithm would be deployed directly onto CCTV traffic cameras to detect car crashes and provide real time alerting.

SOURCING DATA

The two challenges in supervised learning are sourcing and labelling data. Given the complexity and time requirements necessary to collect training data on car crashes directly from CCTV cameras we chose to search through the world's largest source of recorded video, Youtube, for videos containing car accidents. In our search to narrow towards videos with car crashes we found that Ankit Shah et al. worked on a similar problem at the Carnegie Mellon University, Language Technologies Institute[4]. Upon request, they graciously allowed us access to the data sources they utilized which contained a set of 1416 video ids and 518,256 extracted video frames totaling 45GB and 5.2 hours containing compilations of car crash example.



(a) Number of objects by categories (b) Number of usual/unusual tracks (c) Number of objects by sizes

The average frame rate per second for this dataset is 28 fps. The number of objects by category is very skewed towards cars given the nature of the problem. This makes detecting impacts between cars and 'other' or 'other' and 'other' harder to detect given the lack of training examples. In each video the accident starts, on average, 3.69 seconds in. This gives 144,653 frames, ~28% of the dataset, that do not have an accident. The challenge of this dataset was to correctly label each frame. All videos contained one or more accidents but the frames that made up each video were not individually labeled.

DATA LABELLING

We used three approaches to label the training data:

Google Vision API: A Semi-supervised Approach

To help with the locating frames that we suspected had an accident we utilized the Google Vision API. First, the frames were loaded into a Google Cloud Storage bucket and then the Vision API was used to detect relevant labels for ~115,000 images. Of those, 9,543 frames were identified as an accident (having labels such as crash, glass or event). To get a baseline for precision (True Positive Rate), we then manually annotated those 9,543 images on their actual classification of collision or not. Since we realized the Google Vision API had likely missed a fair amount of false negative images we took a randomized sample of 10,000 images labeled as FALSE and annotated these images manually ourselves. Given the total set of images manually annotated consisted of around 20,000 frames we were able to get a good mixture of the variations in lighting schemes, location, and weather in which an accident could be classified.

Mechanical Turk - Labeling Data At Scale

Since manual labeling of frames is a time intensive task we limited our own labeling efforts to 20,000 frames to allow adequate time for model selection, evaluation, and improvement. This meant we were left with a remaining 95,000 unlabeled frames which likely still had valuable features our models could learn from. In order to assess the potential of labeling these remaining frames through crowdsourcing efforts we took a small sample of 150 frames and opened up an image classification task split across 3 various Amazon Mechanical Turk workers. Results were mixed with an average of around ⅔ of workers identifying the correct label for an image and an overall accuracy across all images of 68%. Given this low accuracy we determined that this would not be a scalable solution towards building valuable training data across the remaining frames. Further evaluation of the Amazon Mechanical Turk revealed that the low accuracy is due to the fact that MTurk randomly assigns available workers to each image. While this process can be highly effective for simple image classification efforts such a solution fails to consider the temporal relation within our dataset. Given that an accident can occur over a set of frames, it is important that the same individual is assigned to tag all frames within a video where an accident occurs as an accident can become less visible to the human eye as cars move out of frame or become mostly blocked by other traffic. This experiment

revealed that while crowdsourcing would help us expand label quantity we would require a customizable labeling solution beyond what MTurk offers to create quality labels at scale.

THE MODELS - RESNET, VGG AND INCEPTION

Using a transfer learning approach, we used three platforms to train our models: DIGITS, Tensorflow, and Nvidia's Transfer Learning Toolkit.

Data Splitting: Frame-wise splits outperform video-wise splits

We use 80% of the data for training and equally distribute the rest between validation and test sets. From an image classification perspective, the methodology used for splitting can play an important role. Originally, we split the frames across the three sets (i.e. train, validation and test). We noticed that the in-the-wild results for the models were much lower than test accuracies. We believe that the following method would result in better memorizing than generalization as the model would have possibly seen a similar setting in the training set (few frames earlier).

The second approach was to split the frames video-wise, resulting in all frames of a particular video being allocated to one of the three sets. Using this approach, the in-the-wild performance is in line with test accuracies. The performance of the models deteriorated substantially on most metrics. We believe the limited size of the dataset (trained on ~250 unique videos) could be the reason behind the lackluster performance. The recall rates improved marginally.

We also use an "in-the-wild" dataset to validate the performance of the models trained using framewise data splits. Frames were extracted from a CCTV compilation video from Thailand at 1fps and 2fps (more manageable than 28fps). At 1fps and 2fps the test data is made up of 66% collisions and 34% no collision. We build a baseline for this video parsing these frames through the Vision API. The performance of the baseline (Vision API) gives us 51% precision and 46% overall accuracy. Error analysis suggests the inability to classify motorcycle accidents well.

TRAINING AND VALIDATION

We used Tensorflow to train over the Inception_v3 and Mobilenet pretrained models. NVIDIA's Transfer Learning Toolkit (Tensorflow back-end) was used to train on ResNet-18, ResNet-50, and VGG-16. Finally, we used a pre-trained GoogleNet on the DIGITS platform. We found better results unfixing the lower layers (as compared to fixing the lower layers and only training the last layer). We summarize the performance of all the models here:

Model performance using frame-wise splitting of data

	Test	In-the-wild Performance			
	Accuracy	Accuracy	Precision	Recall	F1 score
ResNet-18	90	56	79	46	58
ResNet-50	91	63	79	59	68
VGG-16	90	59	89	44	59
GoogleNet	96	64	85	55	67
Inception V3	83	65	80	63	70

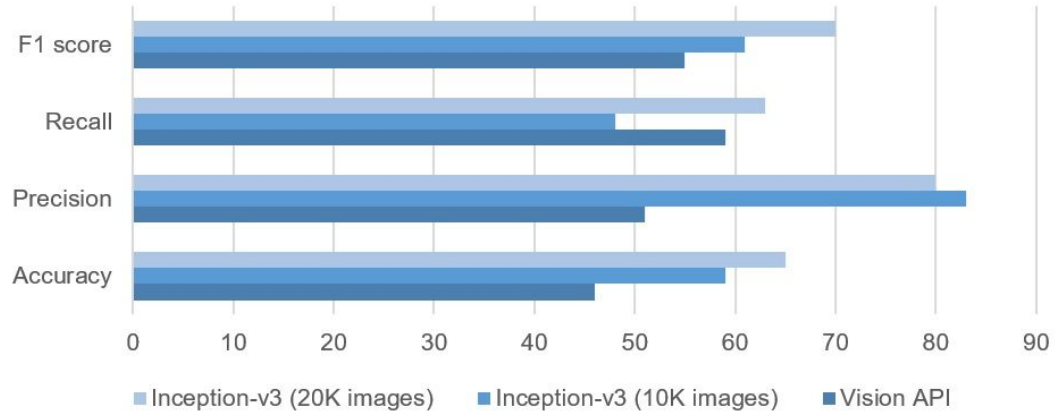
INCEPTION V3: Highest F1 Score In The Wild

Our first model builds on the Inception-v3 architecture, the first runner-up for image classification in ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2015. A convolution neural network with 42-layers, the architecture offers accuracy at the expense of latency. We add a softmax layer on the top of the architecture in order to classify images into two categories - collision or no-collision.

Data Drives Accuracy

The importance of the training data is evident given the difference in performance between 10,000 images and 20,000 images. The performance improvement is evident with a 6% increase in overall accuracy and 9% improvement in f1 score:

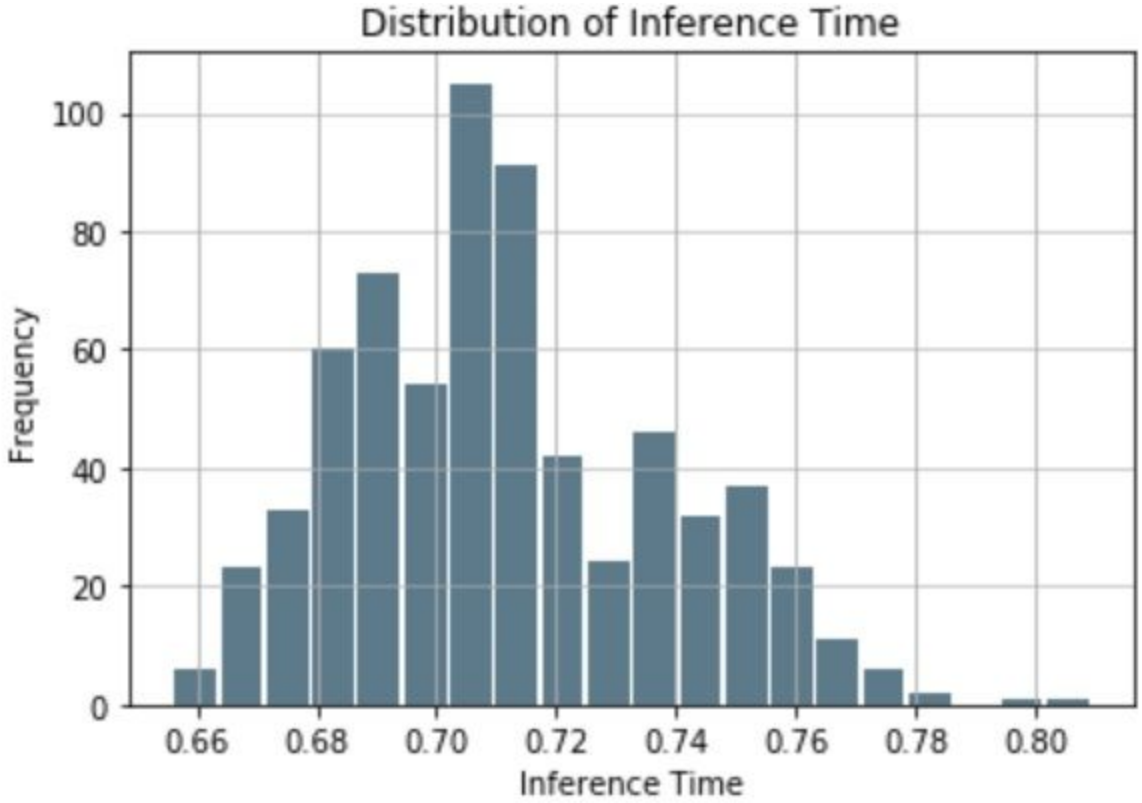
Improvement in performance due to transfer learning and additional data



We expect that with additional time or a more effective crowdsourced labeling solution we could further expand the training set to continue to see improvements in both accuracy and f1-score.

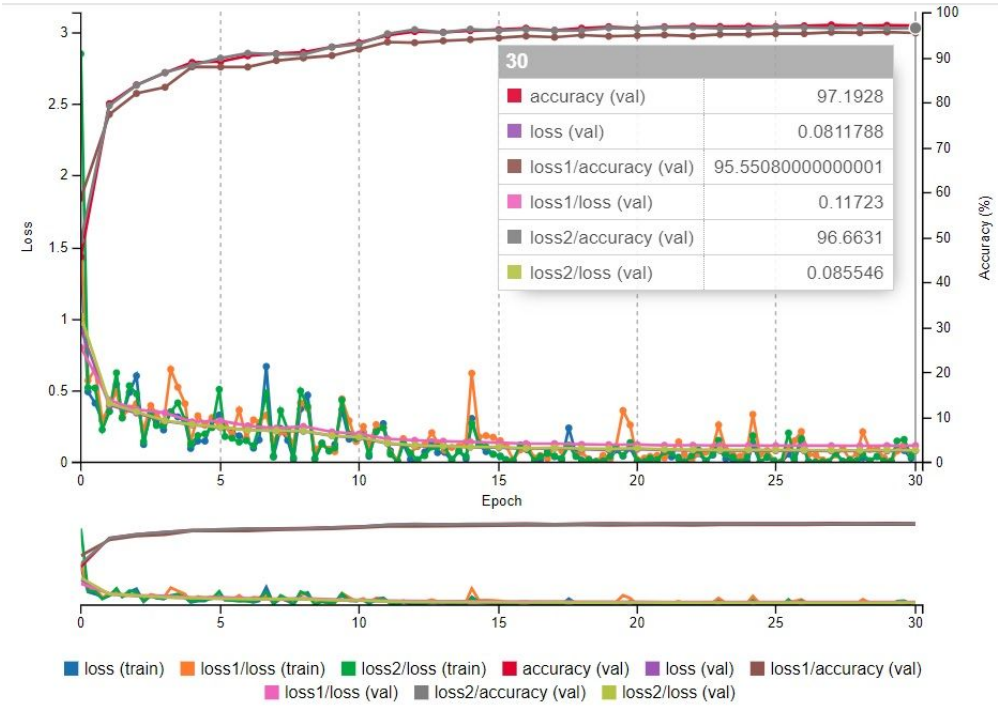
Inference Time - the weakest link

Latency plays an important part in the current scenario, given the nature of the solution. The median inference time of the model built on the inception-v3 architecture is ~0.7 seconds per frame. This implies that the fastest frame rate the model can be fed with is 1fps. As road accidents occur in fractions of a second, there is a risk of the detection algorithm missing an event due to the low inference time.



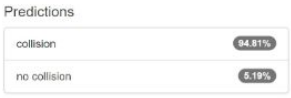
NVIDIA DIGITS: GoogleNet Retrained Across All Layers

The second model showcased is built on the DIGITS framework. It re-trains a Googlenet architecture, modifying the weights on all the layers. While this increases training time, there is a substantial improvement in validation accuracies. We see that accuracy and loss plateaus after 15 epochs and there is marginal learning gains post that.



Performance

The test accuracy of the re-trained Googlenet (i.e. Inception v1) model exceeded all others, with 97% accuracy on the test set (framewise-split). In the wild, the performance seems to be similar to inception v3 in terms of f1-scores. It outperforms Inception-v3 in terms of precision (85% vs. 80%) and lags on recall (55% vs. 63%). Also, we are able to see improvements in the models ability to detect motorcycle accidents (after-impact) more effectively.



Training Time

Due to a smaller dataset, the total training time for 30 epochs was 17 minutes on 2GPUs (P100). The training was much faster (9 minutes) when the lower layers were fixed.

Deployment Benefits

The biggest benefit of the DIGITS framework is the ability to deploy a downloaded caffe model on the Deep Learning SDK offered on the Jetson. As the SDK is written in C++, the rendering speeds are high and real-time accident detection is made possible on the Jetson.

Here is a link of screen recording of real-time accident detection. We simulated the scenario by playing our video and focusing the USB camera on the screen.

NVIDIA TRANSFER LEARNING TOOLKIT (NGC TLT)

The most promising and the least compatible framework was the NVIDIA Transfer Learning Toolkit. While training and evaluation was seamless, the *uff* and *tlt* file extensions are only useful in the Deepstream SDK. Unfortunately, we were unable to evaluate the same. We trained and evaluated the three architectures on the NGC TLT: resnet-18, resnet-50 and vgg-16.

RESNET-18 and RESNET-50

The Resnet architecture won 1st place in the ILSVRC 2015 classification competition with top-5 error rate of 3.57%. The architecture attempts to solve the degradation problem as network depth increases. It does so by using 2-layer (ResNet-18) or 3-layer (ResNet-50) residual blocks.

ResNet Performance

While the test accuracy for both architectures were similar (90%), the ResNet-50 outperformed the ResNet-18 in the wild. Accuracy rates were 7% higher than ResNet-18 (63% vs. 56%) specifically on the back of better recall rates. For ResNet-18, the validation and test accuracy was 93% and 89% respectively. In comparison, the ResNet-50 exhibits a marginally higher test accuracy (91%) and validation accuracy (94%) after 10 epochs.

VGG-16

VGG16 (Simonyan et al, 2014), which achieved 92.7% top-5 test accuracy in ImageNet in 2014, offers the highest precision of 89% in the wild. Although, the conservative nature of the classifier is evident with the lowest recall rate across all models (44%).

ERROR ANALYSIS

With 82% of the collisions involving motorcycles, we believe the in-the-wild performance is a conservative measure of our models. Our original training data comprised of frames labelled as ‘collisions’ by the Google Vision API. Due to the inability of Vision API to detect motorcycle accidents, almost half the labelled dataset did not include motorcycles and a bias was built in the training data. Despite the training data being mostly cars and pedestrians, the precision rate for motorcycle accidents is relatively high. The model accuracy is more in line with expectations from the limited two-wheel vehicle accidents that were in the training set. To help with error analysis, the [test video] was rendered with the prediction and the accuracy score. We highlight some misclassified video frames across different models

Inception-v3: The models are not assessing the aftermath of motorcycle accidents effectively

GoogleNet Unfixed: Accidents near the corner of the frame are not detected effectively

CHALLENGES AND LIMITATIONS

There is a need to improve recall rates and speed up inference time. Our best solution offers a 63% recall rate and an inference time of 0.7 sec/frame. Given below are some of the challenges we may need to overcome:

Need More Data: data augmentation could assist

The number of angles a surveillance camera can be placed is unmeasurable. A single accident can be viewed from a multitude of different angles, depending on how the camera is angled within the 360 degree potential visual field of an intersection. While 20,000 frames allowed us to train our models on various angles, we are merely scratching the surface of the search space. This is possibly one of the reasons that in-the-wild performance suffered in the absence of obvious markers such as smoke, debris or structural damage. In the current scenario, the model performed well in detecting impact but fell short on identifying the aftermath of an incident. While we attempted an object detection approach, the annotation requirements were costly and time consuming. The low precision from HIT tasks piloted on MTurk was also a deterrent.

DISCUSSIONS AND FURTHER WORK

Decision Support

Given the considerable amount of data needed to improve the model accuracy and recall, there are two scenarios where our preliminary solution can be used to impact real-world results. The first scenario is where a human does not intervene between the model prediction and the alerting of emergency medical services (EMS) to go to the scene. In this scenario accuracy and recall are less important than precision. Having a high precision is more important because you don't want to waste EMS's time traveling to a scene where there is no accident. For example, having an accuracy of 30% and precision of 95% is preferable to an accuracy of 60% and a precision of 70%. With 70% precision, 30% of label accidents will waste the time of EMS. The

second scenario is where a human or a few humans sort through potential accidents and manually alert EMS of an accident. With humans, precision can be more relaxed and accuracy and recall become more important to get a breadth of accident types. Ultimately, having humans to correctly label an imperfect model would help the model get better over time so that it can be implemented automatically with minimal human intervention.

A multi-class approach to accidents: Impact and post-impact classification

Currently, our solution is a binary classifier detecting collisions (and no collisions). As discussed briefly above, we believe that the model performance varies during the different phases of an accident. A three-class approach could help us distinguish model performance during the different phases of a road collision.

Background Subtraction: Focus on what matters

With high precision rates, we believe the model seems to memorize better than generalize. The different camera angles prevent us from applying image mean subtraction at a dataset level. Background subtraction at a video-level should be evaluated for any performance lift.

Video Understanding: Anomaly Detection Using Video Embeddings

As an accident is a sequence of events, there is potential in evaluating the scenario as a sequence of video frames. LSTM and RNNs could be ideally placed in such a scenario. Similar to word vector embeddings, video embeddings could be another approach to reduce frame-wise labelling efforts. We could also explore unsupervised learning to detect anomalies.

CONCLUDING REMARKS

As out-of-the-box API solutions (like Google Vision API) are not effective in capturing domain-level context, our solution clearly outperforms the above-mentioned baseline. Accuracy is merely driven by data. Although crowdsourcing offers us a mean to source accident videos (i.e. Youtube), labelling these frames is a bigger challenge. Semi-supervised and crowdsourcing approaches fare poorly. While a hand-crafted labelled dataset seems to offer high precision rates (through memorization), it fares poorly on recall rates. Ironically, Emergency Response systems would prefer to err on the side of caution (prefer recall over precision). We propose to leverage the solution as decision-support tool improving the effectiveness of teams monitoring vast grids of surveillance cameras. We believe a semi-supervised approach to labelling data is required to build a scalable solution. Also, video understanding is uncharted territory and offer a novel solution. Rather than frame-level classification/detection, deep learning on video-embeddings could help understand accidents (anomaly detection) more intuitively through sequence of frames.

ACKNOWLEDGEMENT

A special thanks to the Shah et al. who saved us a lot of time by sharing their repository of video frames, which formed the backbone of the training data. We would also like to thank Dima Rekesh and Estaben

REFERENCES

1. CADP: A Novel Dataset for CCTV Traffic Camera based Accident Analysis, Shah et al (2018) [here]
2. Rethinking the Inception Architecture for Computer Vision (2015) - Szegedy et al. [here]
3. TensorFlow-Slim image classification model library [here]
4. NVIDIA Transfer Learning Toolkit [here]
5. Road Safety Facts [here]
6. Hit-and-run cases in India [here]
7. Hit-and-Run deaths at all-time high in USA [here]

APPENDIX

1. Demo video of accident detection inference using Inception V3 [here]
2. Demo video of real-time accident detection on the Jetson TX2 development kit [here]