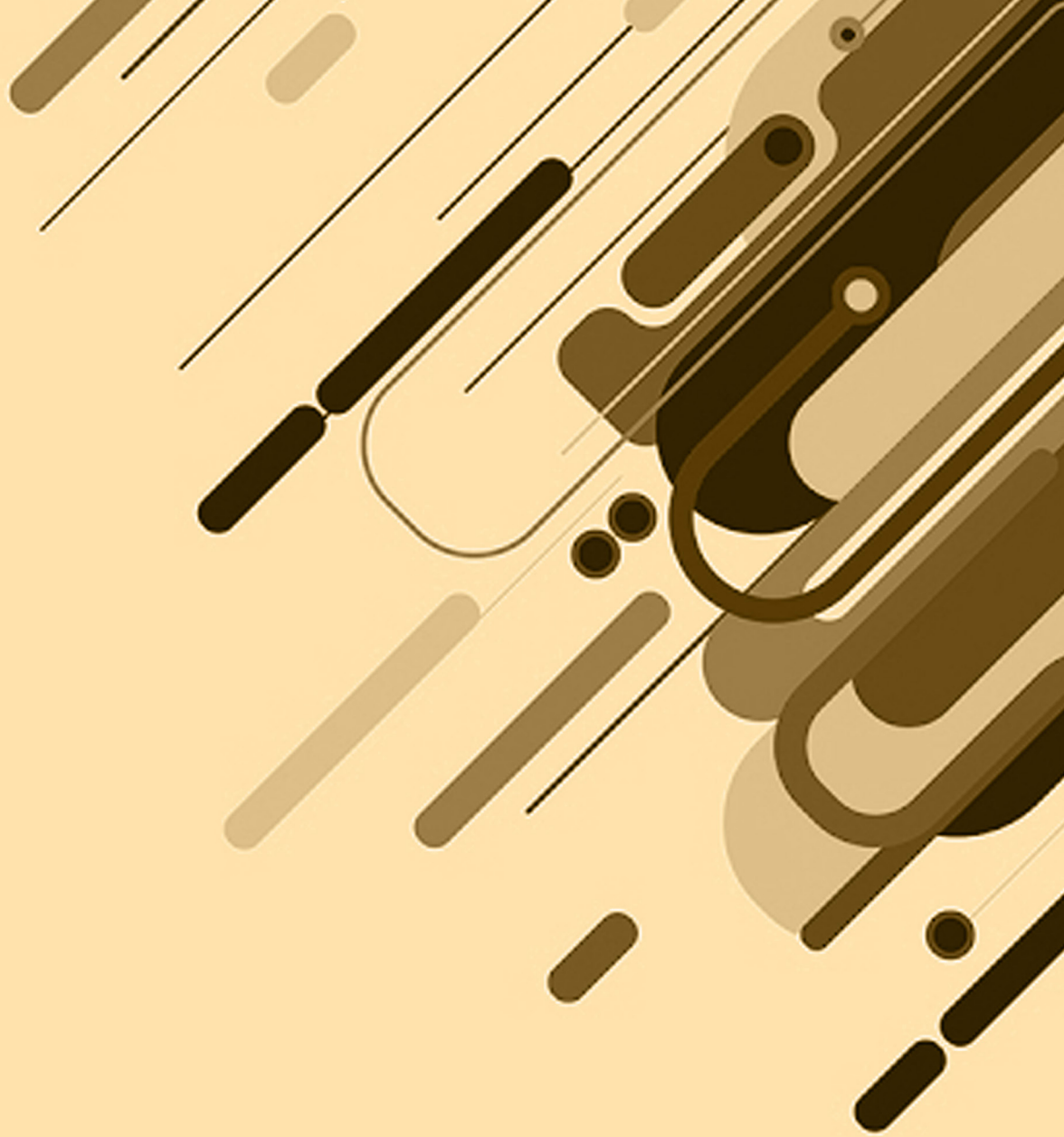


Project Report on

Trigger Automated Ambulance Response System (TAARS)

Abstract

In this report we propose and investigate a nobel end to end method to capture data (Geo-location) of road accidents by identification of spots through a machine learning based image recognition system and hence trigger a request for an ambulance and other related authorities in order to save lives and ensure smooth traffic flow. We aim to have a PWA application and a computer vision setup to capture data of accidents on national and state highways (initial approach),so that we can identify the spots with the occurrence of maximum accidents and their changing pattern on timely basis and trigger automated response. It is designed to throughput and process hundreds and millions of data daily. The system exploits the state of the art, image recognition, image processing and large scale machine learning.

We describe the architecture of the system, the procedure and the time frame we are following to build the system and the challenges we are facing in the due course.We are trying to introduce a process which can response to multiple call to action.

Keywords: *Computer Vision, Data Analysis, Object Recognition, Progressive Web Applications.*

Index

List of figures

Chapter 1
Introduction

Road accidents is a negative externality associated with expansion in road network, motorization and urbanization in the country. Road traffic injuries are recognized, globally, as a major public health problem, for being one of the leading causes of deaths, disabilities and hospitalization, imposing huge socio-economic costs. In case of India, road injuries is one of the top four leading causes of death and health loss among persons of age group 15-49 years.

According to recent reports on Road Accidents,India accounts for highest number of road deaths in the world, according to Geneva based International Road Federation (IRF).India accounts for the 10 percent of the global road accidents with more than 1.46 lakhs fatalities annually.In the 2016,report published by Transport Research wing under Ministry of Road Transport & Highways, Government of India, has revealed that the number of people dying on roads accidents in India has increased on an yearly basis.

The data has further revealing the new recordings with data from previous year shows that in spite of recording fewer accidents in 2016, more deaths have occurred this year as in 2015,in 2015, 1,46,133 people had died in 5,01,423 accidents. Theed that the states of Uttar Pradesh and Tamil Nadu have accounted for maximum number of deaths this year. As per the data cited in the report, the country recorded at least 4,80,652 accidents in 2016, leading to 1,50,785 deaths. The number suggests that at least 413 people died everyday in 1,317 road accidents. Further breaking down the statistics, the data reveals that at least 17 deaths occurred in road accidents in 55 accidents every hour in the given time period. Comparing the accident severity, which is measured as the number of persons killed per 100 accidents, was recorded at 29.1 in 2015 which is lower than 31.4 in 2016. Keeping an eye on the severity of the situation we came up with our problem statement and are trying to site a solution to the same.

Table 1.1: Road Length, Motors Vehicles and Road Accidents (2005-2016)

Year	Road Length (in kms)	Total Number of Registered Motor Vehicles (in thousands)	Total Number of Fatal Accidents (in numbers)	Total Number of Road Accidents (in numbers)	Total Number of Persons Killed (in numbers)	Accident Severity (Number of persons killed by 100 accidents)
1	2	3	4	5	6	7
2005	38,09,156	81,502	83,491	4,39,255	94,968	21.6
2006	38,80,651	89,618	93,917	4,60,920	1,05,749	22.9
2007	40,16,401	96,707	1,01,161	4,79,216	1,14,444	23.9
2008	41,09,592	1,05,353	1,06,591	4,84,704	1,19,860	24.7
2009	44,71,510	1,14,951	1,10,993	4,86,384	1,25,660	25.8
2010	45,82,439	1,27,746	1,19,558	4,99,628	1,34,513	26.9
2011	46,76,838	1,41,866	1,21,618	4,97,686	1,42,485	28.6
2012	48,65,394	1,59,491	1,23,093	4,90,383	1,38,258	28.2
2013	52,31,922	1,81,508	1,22,589	4,86,476	1,37,572	28.3
2014	54,02,486	1,90,704	1,25,828	4,89,400	1,39,671	28.5
2015	54,72,144	2,10,023	1,31,726	5,01,423	1,46,133	29.1
2016	-	-	1,36,071	4,80,652	1,50,785	31.4

Sources:

1. Accidents – State Police Authorities
2. Road Length – Basic Road Statistics, M/o Road Transport & Highways
3. Vehicles – Road Transport Year Book, M/o Road Transport & Highways
4. Data for Road Length and Registered Motor Vehicles is for the financial year upto 2015 & not available for 2016

Note: Road Length is inclusive of all roads constructed under Pradhan Mantri Gram Sadak Yojana and the erstwhile Jawahar Rozgar Yojana.

Table 1
Road Length, Motor Vehcles and Road Accidents (2005-2016)

ROAD ACCIDENTS IN INDIA, 2016

17 deaths on India's roads every hour, Chennai and Delhi most dangerous

Official data show more people died on Indian roads in 2016 than in 2015; UP and Tamil Nadu accounted for the largest numbers of fatalities

BIG NUMBERS

IN THE YEAR		EVERY DAY		EVERY HOUR	
ACCIDENTS	DEATHS	ACCIDENTS	DEATHS	ACCIDENTS	DEATHS
4,80,652	1,50,785	1,317	413	55	17

FEWER ACCIDENTS, MORE DEATHS

	2015	2016	% change
Accidents	5,01,423	4,80,652	−4.1
Killed	1,46,133	1,50,785	3.2
Injured	5,00,279	4,94,624	−1.1
Severity*	29.1	31.4	7.9

*Accident Severity is the number of persons killed per 100 accidents

HIGHWAYS NOT THE BIGGEST KILLERS

Road type	Share of accidents	Share of accident deaths	Share of persons injured
National Highways	29.6	34.5	29.6
State Highways	25.3	27.9	25.8
Other roads	45.1	37.6	44.6

All figures in per cent

SPEEDING BIGGEST PROBLEM, MOBILES AN ISSUE TOO

SPEEDING caused 66.5% of all road accidents and 61% of deaths

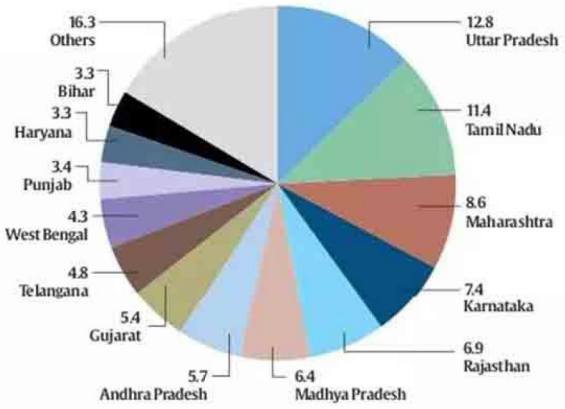
OVERTAKING caused 7.3% of all road accidents and 7.8% of deaths

INTAKE OF ALCOHOL/DRUGS caused 3.7% of all road accidents and 5.1% of deaths

TALKING OVER MOBILES caused 4,976 accidents, 2,138 deaths and 4,746 injuries

84%

DEATHS IN 13 STATES, UP ON TOP

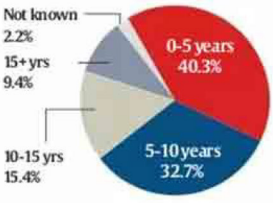


SPEEDBUMPS & POTHOLES KILLED 15 EVERY DAY

SPEEDBREAKERS CAUSED
9,583 road accidents and 3,396 people killed

POTHOLES LED TO
6,424 road accidents and 2,324 fatalities

NEWEST VEHICLES IN MOST ACCIDENTS



Source: Road Accidents in India, 2016, published by Transport Research Wing, Ministry of Road Transport & Highways, Government of India. Compiled on the basis of information collected from the police headquarters of various states and Union Territories and Million Plus Cities through specific designated nodal officers — DGP/ADGP, Crime; ADGP, Traffic; Director, State Crime Records Bureau — in the 17-item Asia Pacific Road Accident Data (APARD) format.

DELHI, CHENNAI ROADS MOST DANGEROUS

CHENNAI: LARGEST NUMBER OF ROAD ACCIDENTS, 7,486

DELHI: LARGEST NUMBER OF ROAD ACCIDENT DEATHS, 1,591

CHENNAI: LARGEST NUMBER OF INJURED PERSONS, 7,349

LUDHIANA: HIGHEST ACCIDENT SEVERITY, 69.9

50 CITIES WITH MILLION-PLUS POPULATIONS ACCOUNTED FOR

- 18.7% of all road accidents
- 11.8% of all road accident fatalities
- 16.7% of all persons injured in road accidents

CITIES WITH THE MOST DANGEROUS ROADS

Cities	No. of accidents
Chennai	7,486
Delhi	7,375
Bengaluru	5,323
Indore	5,143
Kolkata	4,104
Mumbai	3,379

Figure 1

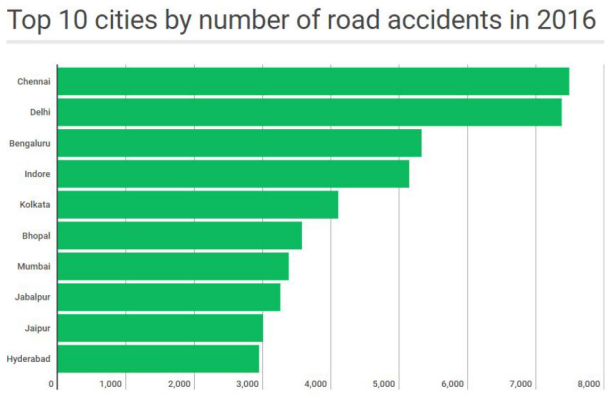


Figure 2
Top 10 cities by number of road accidents in 2016

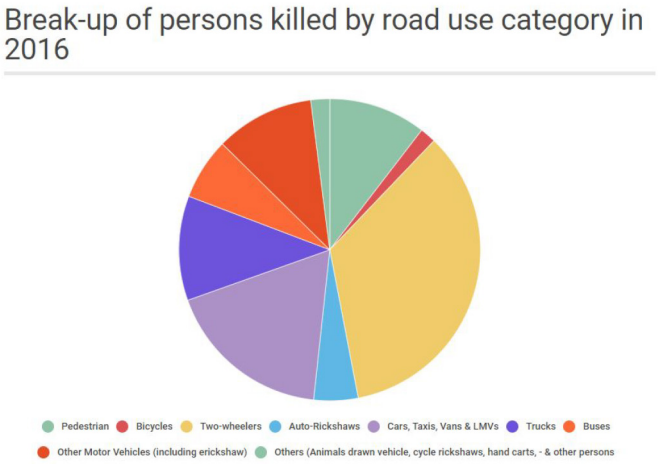


Figure 3
Break-up of person killed by road use category in 2016

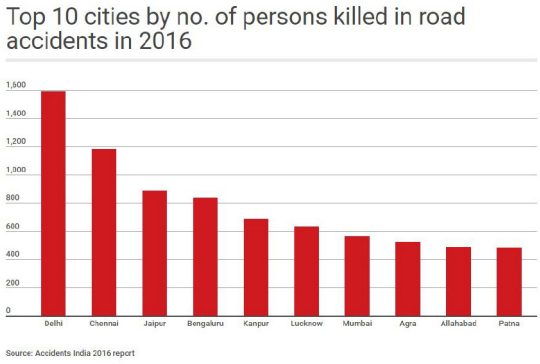


Figure 4
Top 10 cities by number of person killed in road accidents in 2016

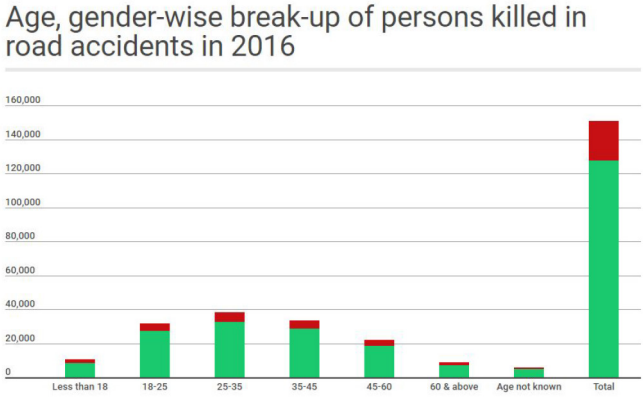


Figure 5
Age, gender-wise break-up of person killed in road accidents in 2016

1.1 Problem Identification

The emergency is the life threatening situation between Life and Death where every second counts and it requires immediate response to save one's life. In such situations, the speedy arrival of an emergency ambulance with paramedics could mean the difference between life and death for a seriously injured or ill person.
The pre-hospital care during life-threatening situations is a neglected issue in India. Although emergency medical services have improved in India by providing common emergency ambulance number 102/108, the practice of emergent care has remained stagnant, with few private hospitals not admitting emergency cases to avoid dealing with medico-legal formalities during emergencies. Many of the medical emergencies like heart attack, stroke, and accidents recovery depend upon the how quickly you get medical care. The lack of medical attention at the time of trauma is leading to nearly 27% deaths in India annually. Thus, India requires better emergency medical services to survive and meet the growing emergency counts.
According to experts, thousands of these accident victims could have been saved if timely medical intervention were available to them. Thus, the need of quality EMS (Emergency medical services) in India is an unmet need for the masses.

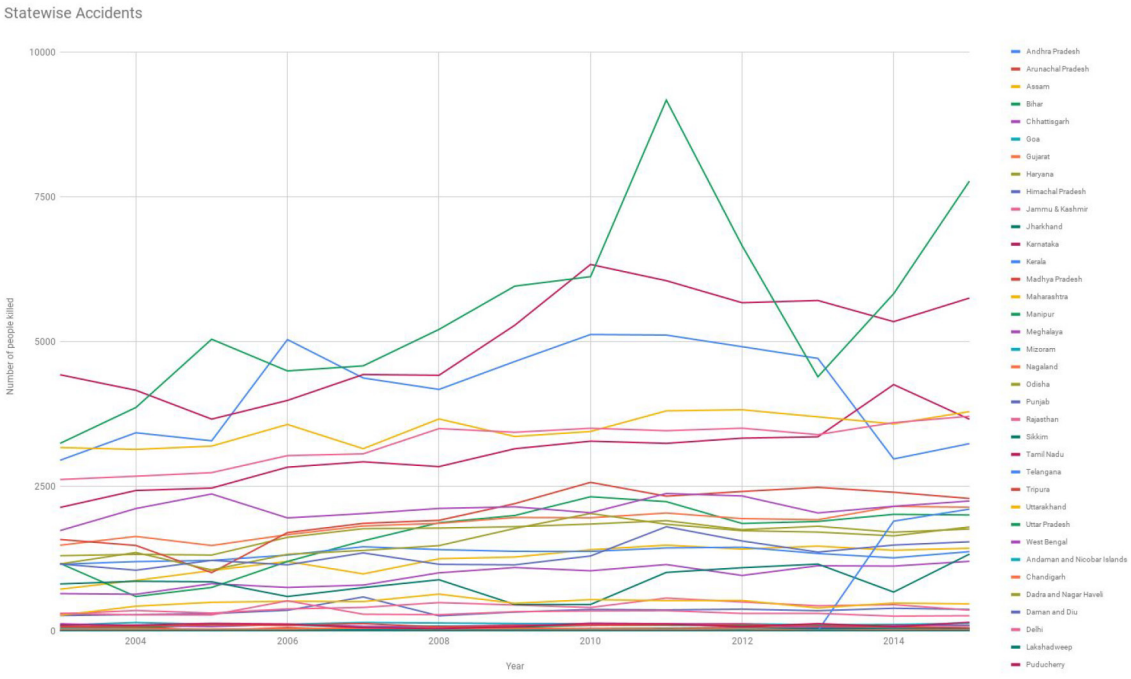


Figure 6
Statewise Accidents 2003 - 2015

1.2 Solution Proposed

A centralized Progressive Web App based application to capture data (Geo-locations) of accidents on national and state highways (initial approach) and identification of spots and its changing patterns based on timely basis. The ultimate aim to trigger a request for an ambulance and the other related authorities as soon as possible, in order to save life and ensure smooth traffic flow.
Aiming to automate the entire process based on the data collected over the period on time and by using the current existing data sets. A Machine Learning based image recognition approach to identify accidents and trigger request.

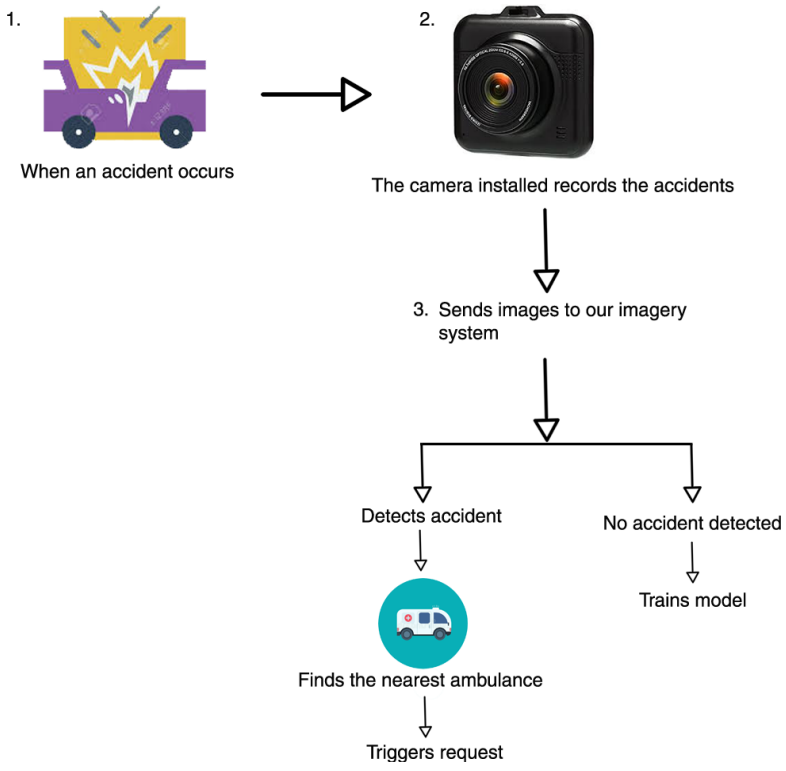


Figure 7
Process flow of the application

1.3 Replacing the traditional approach

For every accident that takes place on a highway,there are two fundamental obstacles between the ambulance reaching the accident spot on time.

1. The location where the ambulance is stationed from the accident spot.
2. The lack of an automated triggering system that can calls an ambulance,which the police authorities do in traditional case.

Our solution solves these two problems using approaches like **Computer Vision, Data Analysis** and **Machine Learning** by :

1. Analysing the data of past accidents taking location (latitudes and longitudes) and climatic conditions as parameters,we can extract a cluster of locations where accidents occur frequently and depending on that,the ambulances can be stationed.
2. Applying image recognition on live video feed from highways,our machine learning model can understand and detect the collision of cars in real time and trigger a call to the ambulance immediately,thus removing the intermediate step of the polic authorities reaching the spot and calling the ambulance.

Chapter 2
Application Overview

Whenever an accident is reported on any of the state or national highways the police is reported about the accident. They arrive at the scene and calls for an ambulance using our application. The application locates the nearest and the quickest possible ambulance that can reach the spot. This functionality uses the data of hospitals provided by the Government. Based on the location of the accident, the nearest hospital is identified by using the Google Maps API (Geolocation). Our algorithm decides on the data returned from the API which action is to be performed that is either call the stationed ambulance or the Hospital.

The data sent to our server is analysed and acts as a feed to the image processing and recognition, and also helps in locating accident hotspots by measuring the number of frequency of accidents at the given geo-location.
Also, we are including a timestamp to the data set, hence we can find the changing pattern of accident hotspots, and also other external factors can be identified. Like the change in hotspots due change in seasons.

2.1 Technologies Used

2.1.1 Programing Language : JavaScript, Python

Why Javascript?

The solution is built as a progressive web app and to access all of the native features like camera and microphone of a mobile device, Javascript is the only solution.

Why Python?

Python makes it easier to process images and make machine learning models using its highly abstract API and libraries like PIL, TensorFlow and YOLO.

2.1.2 Database : Firebase (NoSQL)

Why NoSQL?

NoSQL databases are great for scalability as they don't have a fixed schema, thus different parameters of detection of accident can be fed off instantly based on the complexity of accident making the solution truly scalable and accurate.

2.1.3 Data Analysis & Visualization: Tableau, Python, MS-Excel

Why Tableau?

Tableau has a mapping functionality and is able to plot latitude and longitude coordinates The built-in geocoding allows for administrative places (country, state/province, county/district), postal codes etc to be mapped automatically.

Why Python?

Python has beautiful and easy to use libraries to plot charts and graphs.

Why MS-Excel?

To sanitize the vast amount of accident data and analyse it.

2.1.4 Image Recognition framework: Tensorflow, Open Framework YOLO v2 based on OpenCV

Why TensorFlow?

TensorFlow is an open source Machine Learning library which helps to build Machine learning models easily.

Why YOLO?

YOLO is a python based library that classifies objects from live videos at 30 fps(Frames per Second).

2.1.5 Datasets Used: data.gov.in

2.1.6 Illustrations: Adobe Photoshop, Adobe Illustrator

2.1.7 User Side Application - Progressive Web-Based Application for monitoring the system.

Why a Progressive Web Application?

The reason for using a progressive web application is, it makes the entire system platform independent. The entire user side application can run on any device be it Android, iOS etc., running a Google Chrome Browser (which comes by default in most of the devices). This increases the accessibility and doesn't occupy much of the memory space.

2.2 Basic concepts and methodology of object detection

Vehicle detection using these machine learning and computer vision techniques.

1. Linear SVM
2. HOG(Histogram of Oriented Gradients) feature extraction
3. Color space conversion
4. Binning
5. Sliding Window

2.2.1 Linear Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. The learning of the hyper-plane in linear SVM is done by transforming the problem using some linear algebra.

2.2.2 Histogram of Oriented Gradients

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image.

2.2.3 Color space conversion

Color space conversion is what happens when a Color Management Module (CMM) translates color from one device's space to another. Conversion may require approximations in order to preserve the image's most important color qualities.

2.2.4 Binning (Data and Space)

Data binning or bucketing is a data pre-processing technique used to reduce the effects of minor observation errors. The original data values which fall in a given small interval, a bin, are replaced by a value representative of that interval, often the central

value. It is a form of quantization.

2.2.5 Sliding Window

A procedure is proposed for processing images and detecting objects using a sliding window mode and allowing for an effective realization in terms of the computational complexity and the quality of detection.

2.3 Object detection code

2.3.1 Collecting Data

```
# Get image file names
images = glob.glob('./training-data/*/*/*.png')
cars = []
notcars = []
all_cars = []
all_notcars = []

for image in images:
    if 'nonvehicle' in image:
        all_notcars.append(image)
    else:
        all_cars.append(image)

# Get only 1/5 of the training data to avoid overfitting
for ix, notcar in enumerate(all_notcars):
    if ix % 5 == 0:
        notcars.append(notcar)

for ix, car in enumerate(all_cars):
    if ix % 5 == 0:
        cars.append(car)

car_image = mpimg.imread(cars[5])
notcar_image = mpimg.imread(notcars[0])

def compare_images(image1, image2, image1_exp="Image 1", image2_exp="Image 2"):
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3))
    f.tight_layout()
    ax1.imshow(image1)
    ax1.set_title(image1_exp, fontsize=20)
    ax2.imshow(image2)
    ax2.set_title(image2_exp, fontsize=20)
    plt.subplots_adjust(left=0., right=1, top=0.9, bottom=0.)

compare_images(car_image, notcar_image, "Car", "Not Car")
```

2.3.2 Defining utility functions and libraries

```
# a function to extract features from a list of images
def extract_features(imgs, color_space='RGB', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0,
                        spatial_feat=True, hist_feat=True, hog_feat=True):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        file_features = []
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        if color_space != 'RGB':
            if color_space == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif color_space == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif color_space == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif color_space == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif color_space == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else: feature_image = np.copy(image)

        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
        if hist_feat == True:
            # Apply color_hist()
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        if hog_feat == True:
        # Call get_hog_features() with vis=False, feature_vec=True
            if hog_channel == 'ALL':
                hog_features = []
                for channel in range(feature_image.shape[2]):
                    hog_features.append(get_hog_features(feature_image[:,:,channel],
                                        orient, pix_per_cell, cell_per_block,
                                        vis=False, feature_vec=True))
                hog_features = np.ravel(hog_features)
            else:
                hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                            pix_per_cell, cell_per_block, vis=False, feature_vec=True)
            # Append the new feature vector to the features list
            file_features.append(hog_features)
        features.append(np.concatenate(file_features))
    # Return list of feature vectors
    return features

def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                        vis=False, feature_vec=True):
    # Call with two outputs if vis==True
    if vis == True:
        features, hog_image = hog(img, orientations=orient,
                                  pixels_per_cell=(pix_per_cell, pix_per_cell),
                                  cells_per_block=(cell_per_block, cell_per_block),
                                  transform_sqrt=False,
                                  visualise=vis, feature_vector=feature_vec)
        return features, hog_image
    # Otherwise call with one output
    else:
        features = hog(img, orientations=orient,
                       pixels_per_cell=(pix_per_cell, pix_per_cell),
                       cells_per_block=(cell_per_block, cell_per_block),
                       transform_sqrt=False,
                       visualise=vis, feature_vector=feature_vec)
        return features
```

2.3.3 Heatmaps

```
def get_heatmap(bboxes):
    threshold = 1
    heat = np.zeros_like(output_image[:,:,0]).astype(np.float)
    heat = add_heat(heat, bboxes)
    heat = apply_threshold(heat, threshold)
    heatmap = np.clip(heat, 0, 255)
    return heatmap

def show_images(image1, image2,  image1_exp="Image 1", image2_exp="Image 2"):
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))
    f.tight_layout()
    ax1.imshow(image1)
    ax1.set_title(image1_exp, fontsize=20)
    ax2.imshow(image2, cmap='hot')
    ax2.set_title(image2_exp, fontsize=20)
    plt.subplots_adjust(left=0., right=1, top=0.9, bottom=0.)

heatmap1 = get_heatmap(bboxes1)
heatmap2 = get_heatmap(bboxes2)
heatmap3 = get_heatmap(bboxes3)
heatmap4 = get_heatmap(bboxes4)
heatmap5 = get_heatmap(bboxes5)
heatmap6 = get_heatmap(bboxes6)
show_images(output_image1, heatmap1)
show_images(output_image2, heatmap2)
show_images(output_image3, heatmap3)
show_images(output_image4, heatmap4)
show_images(output_image5, heatmap5)
show_images(output_image6, heatmap6)
```

2.4 Outcomes

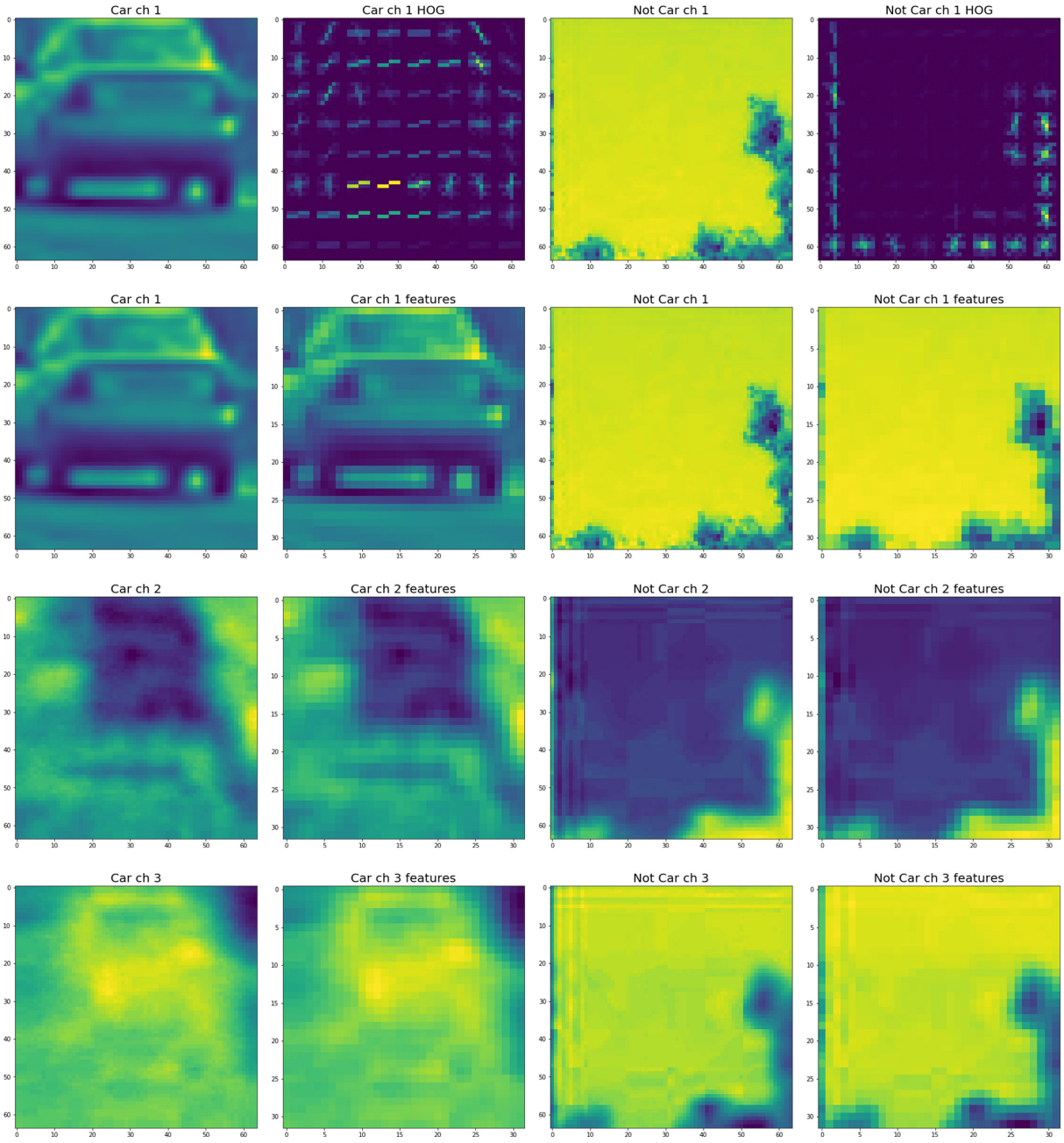


Figure 8
Extracting features

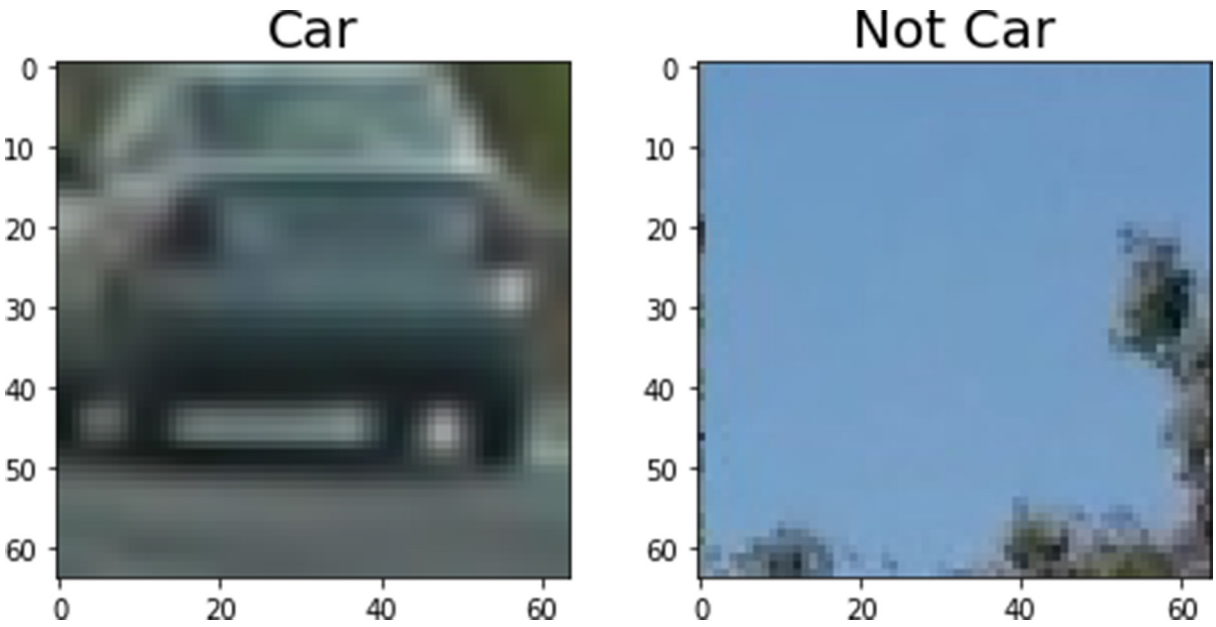


Figure 9
Collecting data and identifying a car

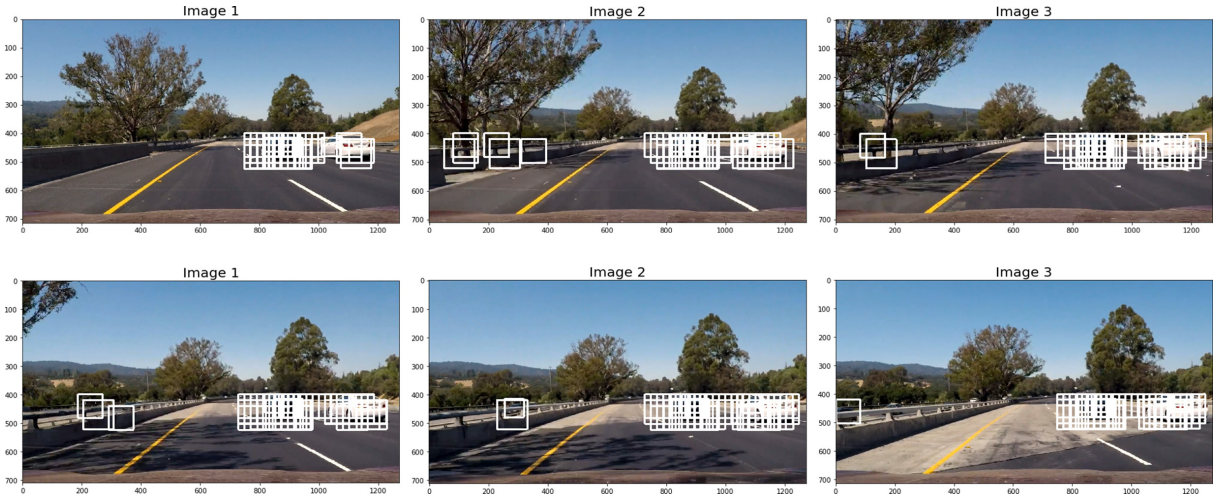


Figure 10
Sliding Window

Figure 11
Final Output

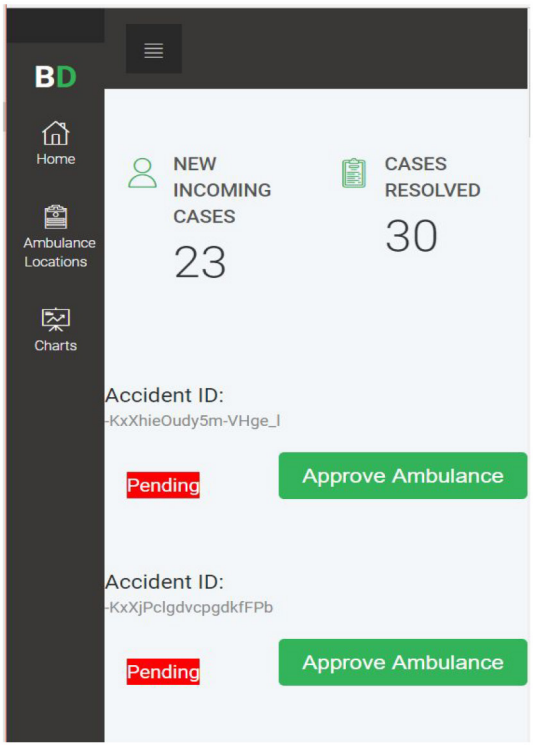


Figure 12
Ambulance Dashboard

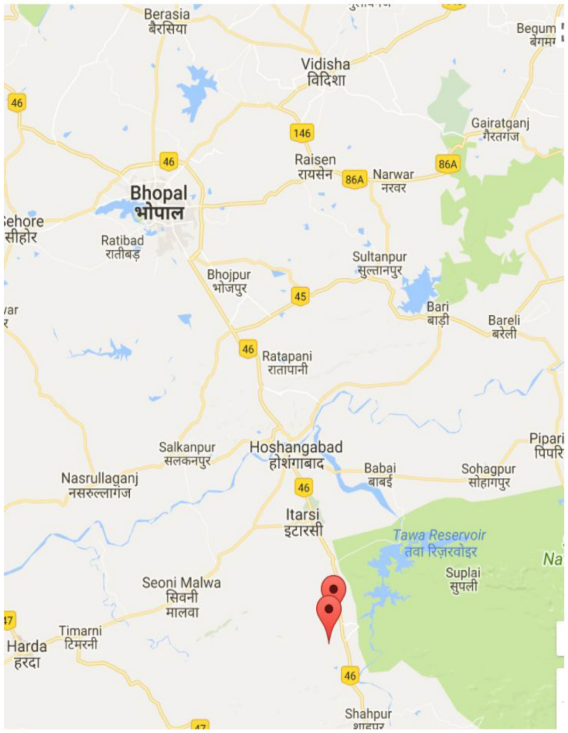


Figure 13
Live Accident Location Tracking

Chapter 3
Timeline

November 2017

The initial idea was framed to solve the above-stated problem of solving road accidents, based on a research journal published by Government of India.

December 2017

Polishing of the idea based on feedback from mentors and multiple approaches to solving the problem was streamlined to using Machine Learning based model for better accuracy and scalability.

January 2018

Progressive web app for the solution was created and tested.

February 2018

Machine Learning model for the solution was built and trained using sample YouTube videos of traffic on roads.

3.1 Current Status

The project is currently ready for the standalone application sending and accepting responses. (Hardcoded). We are working on open source codebase to build the desirous computer vision system for the application. Various open source projects dealt with object detection but none of them pre-trained with accidental data.

3.2 Future Goals and Challenges

Making the entire image recognition system robust and deployable on any central machine or Cloud without using any third party library or codebase.
Making a trained model predict an accident comes up as a next challenge.
Reduction of video data generated by the monitoring system.
Validating the entire application engine and making it compatible with the current video/CCTV feeds from Highways/Roads/Streets etc.

3.2.1 Technology Constraints

1. To make the Machine Learning model truly modular and scalable, the model should be trained in less time which is only possible through GPUs(Graphics Processing Unit), which are very expensive and currently our laptops are not configured for these GPUs.
2. Configuring the model on Cloud is quite an expensive affair.

Chapter 4
References

1. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
2. https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
3. https://www.cambridgeincolour.com/tutorials/color-space-conversion.htm
4. https://github.com/AlabhyaVaibhav/vehicle-detection
5. https://pjreddie.com/darknet/yolo/
6. https://www.tensorflow.org/
7. https://developers.google.com/web/progressive-web-apps/
8. https://firebase.google.com/