Towards partial fulfilment for Undergraduate Degree level Program
Bachelor of Technology in Computer Engineering

A Project Evaluation Report on:

Distracted Driver Detection

Prepared by:

Admission No.	Student Name
U13CO015	**Nilay Kapadia**
U13CO021	**Yash Kanani**
U13CO023	**Manthankumar Vaghasiya**
U13CO032	**Sagar Jogadia**

Class : B.TECH. IV (Computer Engineering) 8th Semester

Year : 2016-2017

Guided by : **R. P. Gohil**

DEPARTMENT OF COMPUTER ENGINEERING

SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY,

SURAT - 395 007 (GUJARAT, INDIA)

Student declaration

This is to certify that the work described in this project report has been actually carried out and implement by our project team consisting of

Sr.	Admission No.	Student Name
1	U13CO015	Nilay Kapadia
2	U13CO021	Yash Kanani
3	U13CO023	Manthankumar Vaghasiya
4	U13CO032	Sagar Jogadia

Neither the source code there in nor the content of the project report have been copied or downloaded from any other sources. We understood that our result grades would be revoked if later it is found to be so.

Signature of the Students:

Sr.	Student Name	Signature of the Student
1	Nilay Kapadia	
2	Yash Kanani	
3	Manthankumar Vaghasiya	
4	Sagar Jogadia	

Certificate

This is to certify that the project report entitled Distracted Driver Detection *is prepared and presented by*

Sr.	Admission No.	Student Name
1	U13CO015	Nilay Kapadia
2	U13CO021	Yash Kanani
3	U13CO023	Manthankumar Vaghasiya
4	U13CO032	Sagar Jogadia

Final Year of Computer Engineering and their work is satisfactory.

SIGNATURE:

GUIDE JURY HEAD OF DEPT.

ABSTRACT

Detection of the distracted driver has become one of the very active research area in intelligent transportation systems. Image Processing and Machine Learning are a great solution to this problem. We will build a model that will first detect if the driver is distracted or not using Image Processing. If the driver is distracted then using Machine Learning we will classify the type of the distraction of the driver, which are texting, talking, drinking, reaching behind, etc. The driver also will be notified about the type of the distraction to alert him to focus on driving.

Keywords: Image Pre-processing, Machine Learning, Human Activity detection

Table of Contents

List of Figures

List of Tables

List of Acronyms

PCA: Principal Component Analysis

SCM: Scalar Vector Machine

EEG: Electroencephalograph

DDAD: Distracted Driver Activity Detection

CNN: Convolution Neural network

ILSVRC: ImageNet Large-Scale Visual Recognition Challenge

ConvNet: Convolution Network

ReLU: Rectified Linear Unit

FC: Fully Connected

VGG: Visual Geometry Group

ResNet: Residual Network

SGD: Stochastic Gradient Descent

GPU: Graphical Processing Unit

CHAPTER 1: INTRODUCTION

More than 5,000 people die each year in vehicle crashes caused by distracted driving, many who were texting or talking on a cell phone behind the wheel. [1] A 2009 study focusing on drivers of larger vehicles and trucks concluded that texting raised the risk of a crash by 23 times compared with non-distracted driving. [1]

As defined in the Overview of the National Highway Traffic Safety Administration's Driver Distraction Program (Report No. DOT HS 811 299), distraction is a specific type of inattention that occurs when drivers divert their attention from the driving task to focus on some other activity instead. There are three main types of distraction:

- Visual distraction: taking one's eye off the road. For example, reading text or watching video while driving.
- Manual distraction: taking one's hand off the wheel. For example, text messaging, eating, or operating the radio while driving.
- Cognitive distraction: taking one's mind off of the driving vehicle. For example, thinking, talking to other passengers, or texting, doing makeup. [2]

Cell phone conversations induce a high level of cognitive distraction, thus reducing the brain activity related to driving by 37% (which might be worse than ingesting alcohol). [2] More importantly, text messaging requires visual, manual, and cognitive attention at the same time, making it the most dangerous distraction. Texting drivers took their eyes off the road for each text an average of 4.6 seconds – which at 55 mph, means they were driving the length of a football field without looking. [2]

Our aim is to determine if a driver is distracted or not, and if he/she is distracted then, the type of distraction should be recognized by the system. In this project, we used Image Processing and Machine Learning techniques for detecting distracted driver.

1.1 Application

The following are some of the applications of identifying distracted driver:

- The Vehicle ad-hoc Networks has emerged as a distinguished branch of wireless communication; that pertains categorically to transportation systems. If the driver is

found distracted, then the system sends out an alert to nearest vehicles using ad-hoc networks.

- The system detects a high risk of the driver, then the driver will be alerted by an audible signal.
- Avoid fraud insurance claims by using recorded footage from a driver monitoring camera and driver activity data.

1.2 Motivation

In last several years, more and more advanced technology has been developed to improve safety like auto emergency braking, curtain airbags, etc. However, irrespective of how advanced technology is, the ultimate reason for the crash are the drivers. If the driver is driving drunk, or get road rage, these technologies are in vain. So, we want to continuously monitor the driver using Image Processing and Machine Learning techniques to detect if the driver is distracted.

1.3 Objectives

With this project, we aim to apply appropriate techniques in Machine Learning to detect if the driver is distracted or not. So, the objectives are as follows:

- Detailed study of Image Processing, with a focus on object detection methods and Machine Learning
- Derive its computational and application issues.
- To discover the strategy to resolve these issues.
- To design a model to detect the distracted driver.
- Implementation of the system to achieve the goal.
- Testing analysis for performance comparison.

1.4 Contribution

The phase wise contribution is as follows:

1.4.1 Contribution (Phase 1)

In the first phase of the project, we studied various methods being used for face detection and work done for detecting distracted driver. We found Viola-Jones algorithm more accurate and suitable for face detection and tested it for sample data. From the test results, we found it useful for certain cases and will implement the improvised algorithm suitable for our application to detect face and hand movements.

1.4.2 Contribution (Phase 2)

In the second phase, we developed an application that will process the video from the camera and use Image Processing if a human is detected, then a notification with the image of the human is sent to the person. This application is analogous to our main objective of the project. We also design the model which uses Image Processing that is going to be used in the processing of the video frame to decide if the driver is distracted or not.

1.4.3 Contribution (Phase 3)

In the third phase, we studied different pre-trained models like VGG16, ResNet50 and InceptionV3 to classify the driver image into different categories which are safe driving, drinking, texting, talking, reaching behind etc. We proposed one VGG-like, classification model in python which have classification layers similar to VGG16 model and was trained and tested on a dataset.

1.4.4 Contribution (Phase 4)

In the last phase, we tried to train our model using pre-trained model vgg16 to classify the images for distracted driver detection. One mobile app on android is created to capture a photo of the driver at regular interval which sends API request to server to classify the image and alert the driver if found distracted.

1.5 Organization of Report

Our report is organized as follows. In Chapter 2, a literature survey is enlisted describing various techniques to detect the distracted driver. Chapter 3 describes Viola-Jones face

detection method. Chapter 4 depicts Convolutional Neural Network and pre-trained model followed by implementation of Viola-Jones face detection method in Chapter 5. Chapter 5 also describes classification model as well as the software implementation of the entire system. Chapter 6 contains the conclusions of this project and the future expectations from these solutions.

CHAPTER 2: LITERATURE SURVEY

In this Chapter, we shall get an in-depth overview of what work has been done on detecting the driver fatigue and distraction. Also, we will discuss the methods proposed for reducing the chances of accidents caused due to distraction.

Early driver monitoring methods were being tested already 20 years ago in the aviation industry [4], [5]. For the pilots, special equipment like EEG was used to measure the heart rate, eye blinks, and stress level. But the car driver is not supposed to wear such extra equipment. So researchers have paid great attention to driver behavioral measures using various techniques such as physiological detection and visual based detection.

A number of studies have shown that driver's behavioral changes due to workload, which can be observed by monitoring the driver directly or by following the vehicle's dynamics. So far possible non-intrusive techniques for detecting fatigue in driver using computer vision focus on methods based on eye and eyelid movements, methods based on head movement and method based on mouth opening.

Paul Viola et al. [6]–[8] have described a face detection framework that is capable of processing images extremely rapidly while achieving high detection rates as a process for training an extremely simple and efficient classifier which can be used as a supervised focus of attention operator and they present a set of experiments in the domain of face detection. Zutao Zhang located the face by using Haar algorithm and proposed an eye tracking method based on Unscented Kalman Filter [9]. Xiao Fan et al. [13] located and tracked a driver's mouth movement using a CCD camera to study on monitoring and recognizing drivers yawning and extracting texture features of driver's mouth corners with Gabor wavelets. R.Wang et al. [9] proposed to track the mouth regions geometric features by a trained well Neural Network. [12]– [14] demonstrate several key techniques in mobile embedded systems.

2.1 Face Detection

We found three techniques to detect a face of the human from the image which is face detection using Haar classifier, geometry base face detection and face detection using Kinect depth data. Detailed description about them is given below.

2.1.1 Face Detection using Haar Classifier

As proposed in paper [15], Adaboost algorithm is a method of statistics-based face detection. Haar classifier with Haar-like features is a major application of Boosting algorithm, which uses contrast values between adjacent rectangular pixel groups to determine relative light or dark regions. Face detection based on Haar-like feature will produce the location information about faces and their organs such as eyes and mouths, which is essential for further facial feature detection. Because the captured image in driving status is not entirely clear, it is an efficient way to locate the eyes and mouth by utilizing approximate position. The following flowchart depicts the algorithm used:

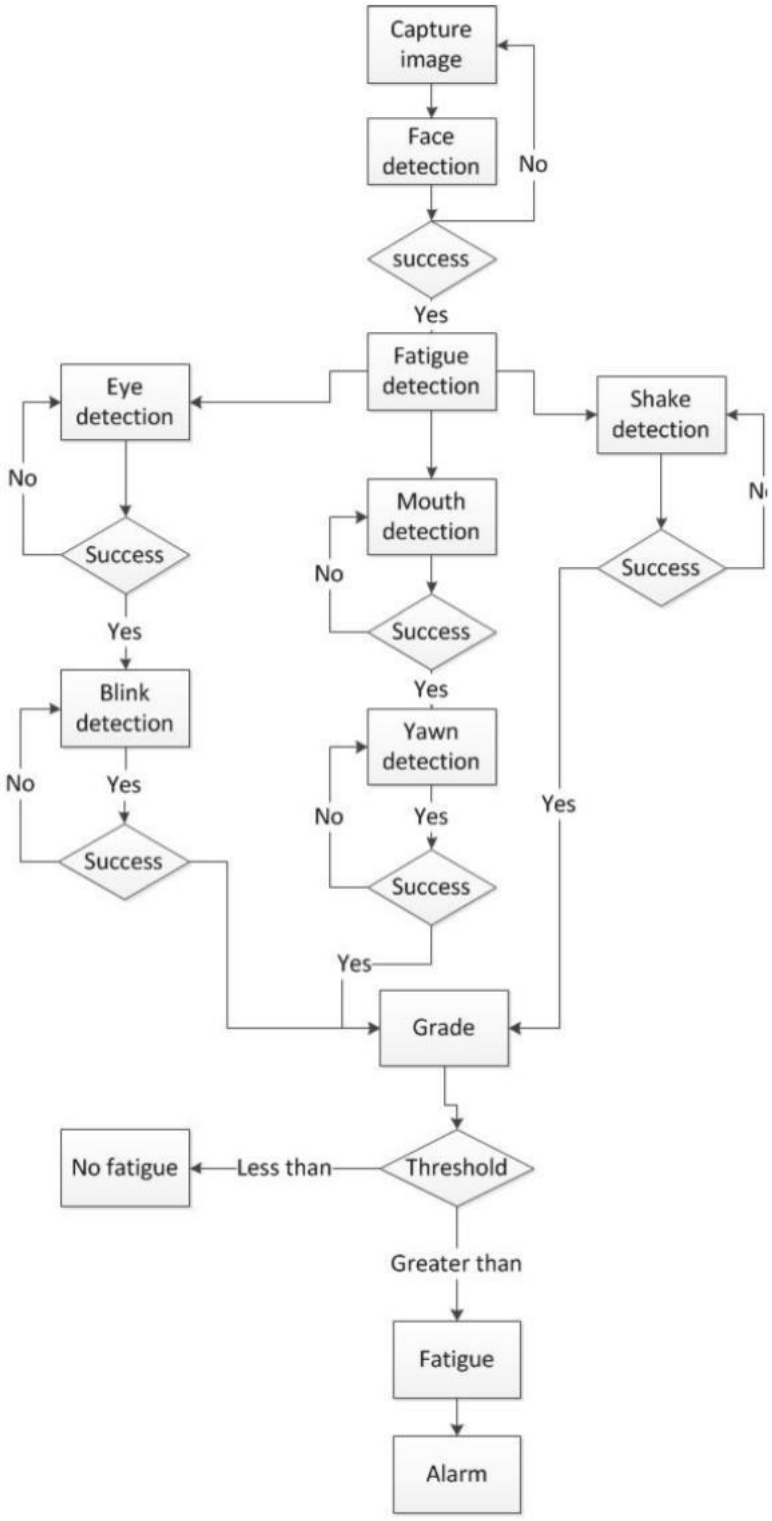


Figure 1: Fusion of Multiple Real-Time Facial Features [15]

2.1.2 Geometry Base Face Detection

For geometry base face detection, Padma Polash Paul and Marina Gavrilova [16] presented a PCA based modeling of the geometric structure of the face for automatic face detection. This method improves face detection rate and limits the search space by using SCM which is one of the best face detection technique for image and video. To model, the geometric structure of face PCA and canny edge detection are used. Fusion of PCA based geometric modeling and SCM method provides higher face detection accuracy, reduces search space and improves time complexity. Another advantage of this method is it is very fast in computation because of the filtering.

Parameter	Haar like Feature base	Geometric base
Precision	High	Low
Execution time	Low	High
Learning time	High	High
Ratio between detection rate and false alarm	High	Low

Table 1: Theoretical comparison of several existing methods in terms of key parameters with feature base face detection [17]

Technique	Merits	Demerits
Geometric base face detection	• Effective approach • Easy to implement	• Low accuracy • More false alarm
Haar like feature base face detection	• Improved feature extraction part • Less false alarm	• High execution time • Complex to implement

Table 2: Pros and Cons of several existing methods [17]

2.1.3 Distracted Driver Detection using Kinect Depth Data

In the paper [2], Kinect sensor is used for detecting distracted driver and recognizing the type of distraction. Their features are based on the depth map data and represent the orientation of the rims of the right arm of the driver. First of all, a background/foreground segmentation is applied to isolate the driver from the driving seat, and background. For feature extraction, only consider the pixels of the right side of the body by removing from the list the pixels on the left-hand side.

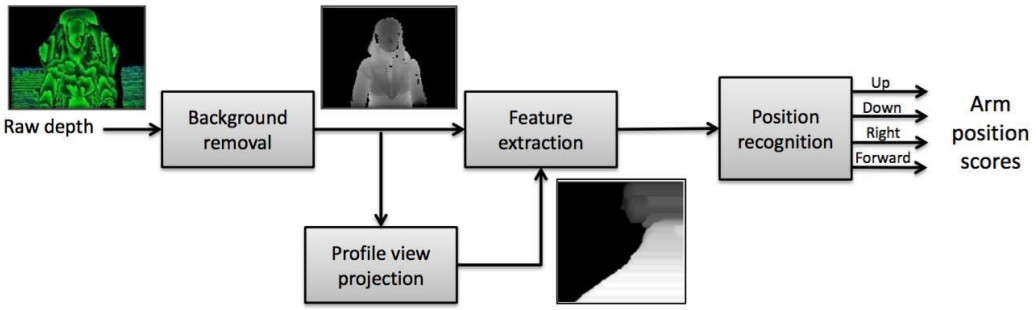


Figure 2: Main blocks of arm position module using the Kinect sensor. [2]

From the Table 1, one can compare which one is better amongst Haar feature and geometry base face detection. Though Haar feature takes more time to detect a face in compare to geometry base face detection, once we detect a face, then we can easily track that detected face, and as in our application, the camera is static, so face detection using Haar feature is the better choice.

2.2 Hand Detection

Now, to detect that driver is not involved in any activities other than the driving, we must detect the hand of the driver to get knowledge about the activities driver is doing at the particular time. We found two different methods to detect a hand.

2.2.1 Position of Hand using Kinect Depth Data

As shown in the Fig. 3, we can use Kinect depth data to detect the position of the hand whether it is up, forward, right or left [2]. From the Kinect, we get Front view image of the driver with the depth data. Using this depth data, profile view is generated.

Among all the extracted features, some are discriminative, some are useless, and some might even be contradictory. Selecting or weighting the features is, therefore, necessary for good classification. Therefore AdaBoost is used, which is a very appropriate tool in this regard. Nevertheless, AdaBoost only solves two-class problems, so a 1-vs-all approach is used: first train four sub-classifiers, each of them specialized for one class (i.e., using as positive examples some samples for a specific class, and as negative examples, some samples from any other class). The estimated position would be the one with the highest value among the four sub-classifiers. The actual output of the module is not exactly the estimated position, but the score of each sub-classifier.

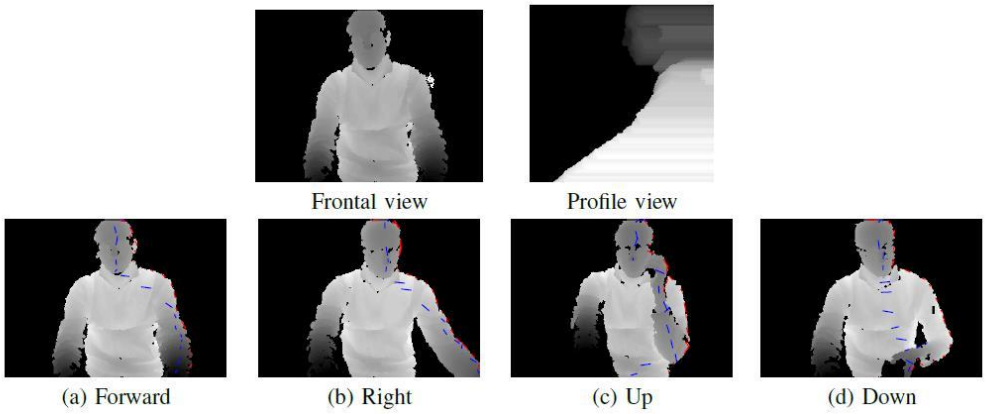


Figure 3: Position of hands using Kinect depth data [2]

2.2.2 Hand Detections in Specified Region

In the vehicle, hand activities also may be characterized by regions of interest. The regions like vehicle steering, gear stick are important for understanding driver activities and any secondary tasks driver is performing. This motivates to detect hand in the particular regions of the image frame only.

Figure 4: Steering and gear stick regions in which hands will be detected [18]

From above two methods, the Kinect solution is most promising, because it generates 3D model of the driver using depth data which is relatively easy to estimate the human pose then in 2D image

CHAPTER 3: FACE DETECTION METHOD

In this Chapter, we shall get an in-depth overview of Viola-Jones face detection method. For detecting the face in real time, we concluded that Viola-Jones face detection algorithm is more efficient, as we have to be precise in detecting whether the driver is distracted or not, and Viola-Jones algorithm helps us to do that.

3.1 Viola-Jones Face Detection Method

The Viola-Jones object detection framework is the first object detection framework to provide object detection in real-time proposed in 2001 by Paul Viola and Michael Jones. It is primarily used for detecting a face.

3.2 Components of The Framework

Feature types and evaluation

The characteristics of Viola–Jones algorithm are:

- Robust-very high detection rate (true-positive rate) & very low false-positive rate always.
- Real time – For practical application at least 2 frames per second must be processed.
- Face detection only

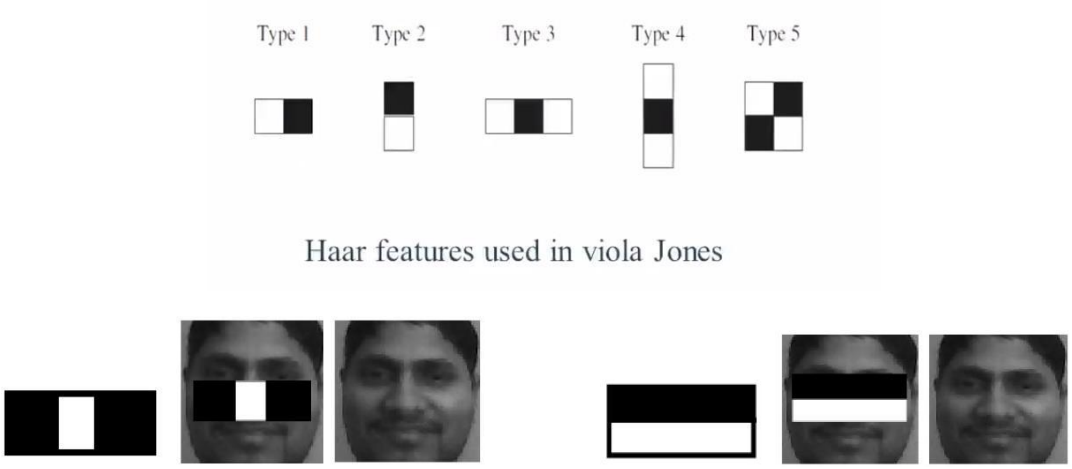
The algorithm has four stages:

1. Haar Feature Selection
2. Creating an Integral Image
3. Adaboost Training
4. Cascading Classifiers

Haar Feature Selection

The Viola-Jones cascade classifier is a face detection algorithm based on Haar-like features. A few Haar-like feature rectangles used in this project are shown in Fig.1. The quality value of feature rectangle is the result of the white section pixel value sum subtracting the black section pixel value sum. The quality value of the feature rectangle is used as the basis for the

face detection, as there are some common properties to human faces like the eye region is darker than the upper cheeks, while the nose bridge region is brighter than the eyes. There is a clear difference between facial color, shape, and location of eyes, mouth, and bridge of the nose, so that can be easily matched.



Haar feature that looks similar to bridge of the nose which is brighter than the eyes is applied into the face

Haar feature that looks similar to the eye region which is darker than the upper cheeks is applied onto a face

Figure 5: Haar feature [3]

Creating an Integral Image

An image representation called the integral image evaluates rectangular features in *constant* time, which gives them a considerable speed advantage over more sophisticated alternative features. It is obtained using few operations per pixel. Because each feature's rectangular area is always adjacent to at least one other rectangle, it follows that any two-rectangle feature can be computed in six array references, any three-rectangle feature in eight, and any four-rectangle feature in nine. The integral image at location (x, y), is the sum of the pixels above and to the left of (x, y), inclusive.

1	1	1
1	1	1
1	1	1

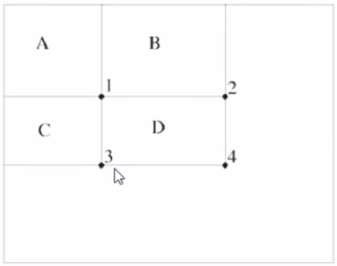
Input Image

1	2	3
2	4	6
3	6	9

Integral Image

Figure 6: Example of the integral image [3]

The calculation can be done using four values at the corners only as shown below in Figure 7.



Sum of all pixels in
D = 1+4-(2+3)
= A+(A+B+C+D)-(A+C+A+B)
= D

Figure 7: More efficient Integral Image calculation [3]

Adaboost Training

In a 24x24 pixel sub-window, there is a total of 162,336[3] possible features, and it would be expensive to evaluate them all when testing an image. It is understood that only few set of features will be useful among all these features to detect a face. Therefore, a variant of the learning algorithm AdaBoost is used in the object detection framework to select the best features and to train classifiers that use them.

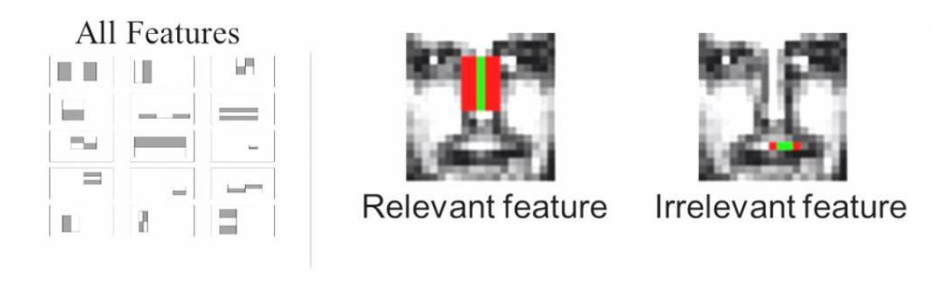


Figure 8: Relevant features and irrelevant features [3]

Adaboost constructs a "strong" classifier as a linear combination of weighted simple "weak" classifiers.

$$h(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x) + \ldots$$

Strong Classifier Weak Classifier

Where, α_i is the weight assigned to h_i that is inversely proportional to the error rate. In this way, best classifiers are considered more.

Cascading Classifiers

Even if an image contains more than one faces, it is obvious that the excessively large amount of the evaluated sub-windows would still be negatives (non-faces). So, the algorithm should quickly discard non-faces and concentrate more on detecting on probable face regions. Therefore, a cascade classifier is used which is composed of stages each containing a strong classifier. Therefore, all the features are grouped into several stages where each stage has a certain number of features. Each stage determines whether a given subwindow is definitely not a face or may be a face. A subwindow is immediately discarded as not a face if it fails in any if the stage.

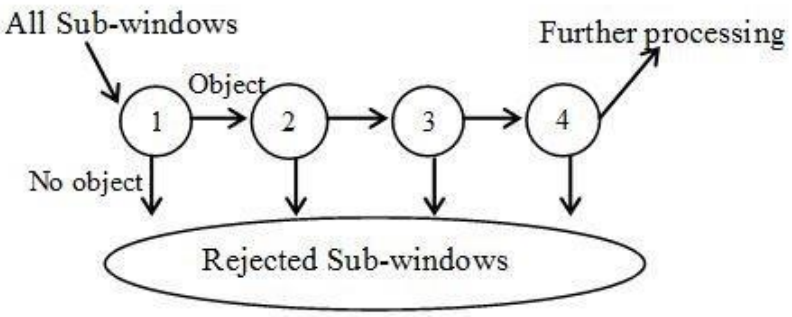


Figure 9: Cascading classifiers [3]

Viola-Jones face detection algorithm forms an integral image and then uses a variant of learning algorithm Adaboost to select the best features and to train classifiers that use them. Then using a cascading classifier, probable faces regions are identified.

CHAPTER 4: CNN AND PRE-TRAINED MODELS

This chapter describes an overview of CNN/ConvNet for image recognition and classification. Pre-trained models like VGG16, ResNet50, and InceptionV3 are also discussed in depth.

4.1 Introduction

CNNs are an often-used architecture for deep learning which is very effective in image recognition and classification. To get good generalization performance, it is beneficial to incorporate prior knowledge into the network architecture. CNNs uses spatial information between the pixels of an image. Therefore, they are based on discrete convolution. They make the forward function more efficient to implement and also vastly reduce a number of parameters in the network.

4.2 The LeNet Architecture

LeNet was one of the very first CNN which helped propel the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1988 [19]. At that time the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc. It was fundamental, in particular, the insight that image features are distributed across the entire image. Convolutions with learnable parameters are an effective way to extract similar features at multiple locations with few parameters. As there was no GPU at that time to help in training, being able to save parameters and computations was a key advantage. The features of LeNet5 can be summarized as:

- use convolution to extract spatial features
- subsample using a spatial average of maps
- non-linearity in the form of tanh or sigmoids
- a multi-layer neural network as final classifier
- sparse connection matrix between layers to avoid the large computational cost

4.3 Layers in ConvNet

ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. ConvNet has neurons arranged in 3 dimensions: width, height, depth.

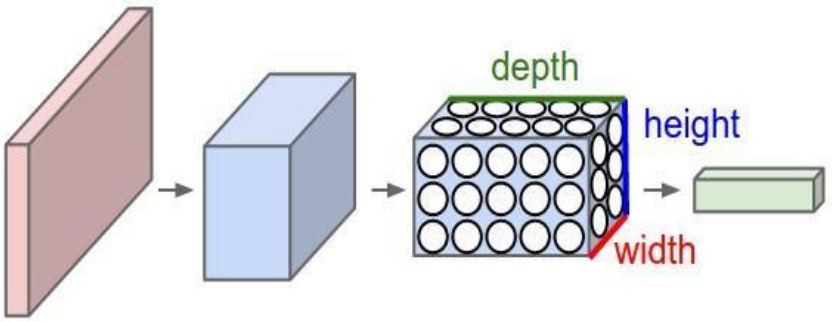


Figure 10: ConvNet arranges neurons in three dimensions [20]

Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. There are four main layers in ConvNet:

4.3.1 Convolution

The Convolution layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The primary purpose of this layer is to extract a feature from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Every image can be considered as a matrix of pixel values. CNNs compare images piece by piece and looks for features. Each feature is like a mini-image which a small two-dimensional array of values

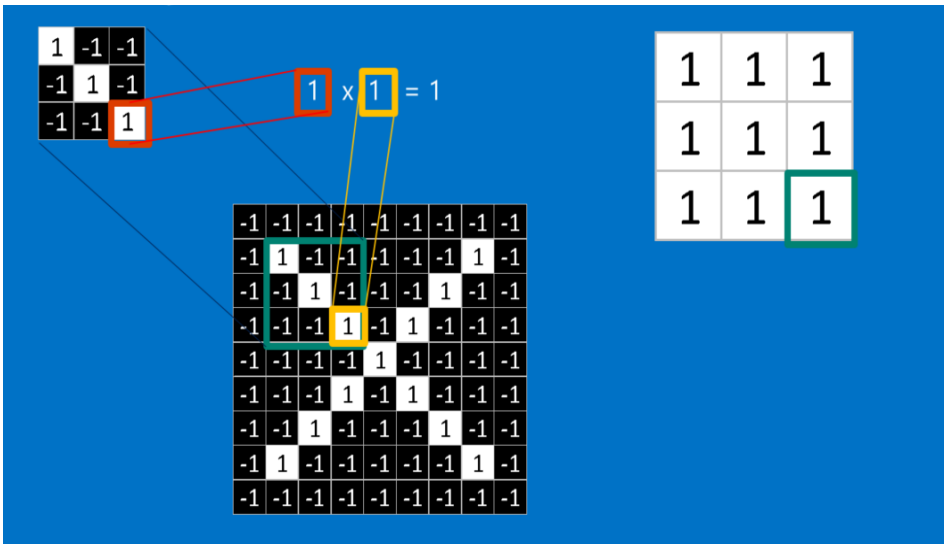


Figure 11: Multiplication of filter with the image [20]

When presented with a new image, CNN tries to find the features everywhere and in every possible position. To calculate the match of a feature to a patch of the image, simply multiply each pixel in the feature by the value of the corresponding pixel in the image and then add up the answers and divide by the total number of pixels in the feature. Every matching pixel results in a 1. Similarly, a mismatch is -1.

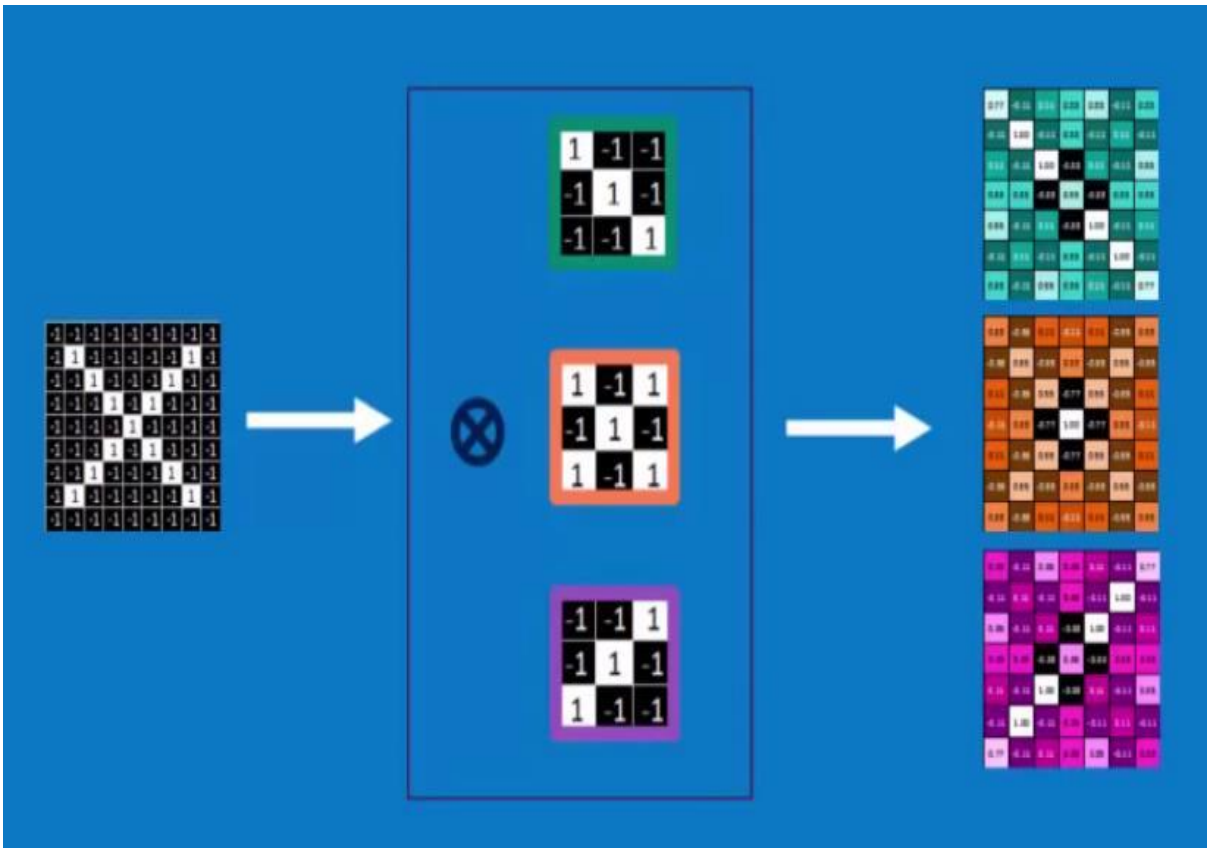


Figure 12: Convolution process [20]

To complete the convolution process, line up the feature with every possible image patch. Take answer from each convolution and make a new 2D array from it. This mapping shows where the feature is found in the image. If the value is close to 1 then it shows strong match and if -1 then it shows strong match for the photographic negative of our feature, and values near 0 show no match of any sort. Now repeat the convolution process for each of the other features. The result is a set of filtered images, one for each feature and this is referred as convolution layer.

4.3.2 Non Linearity (ReLU)

ReLU has been used after every Convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. Convolution is a linear operation as it includes element-wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU. Whenever a negative number occurs, replace it with 0. Other non-linear functions such as tanh or sigmoid can also be used instead of ReLU.

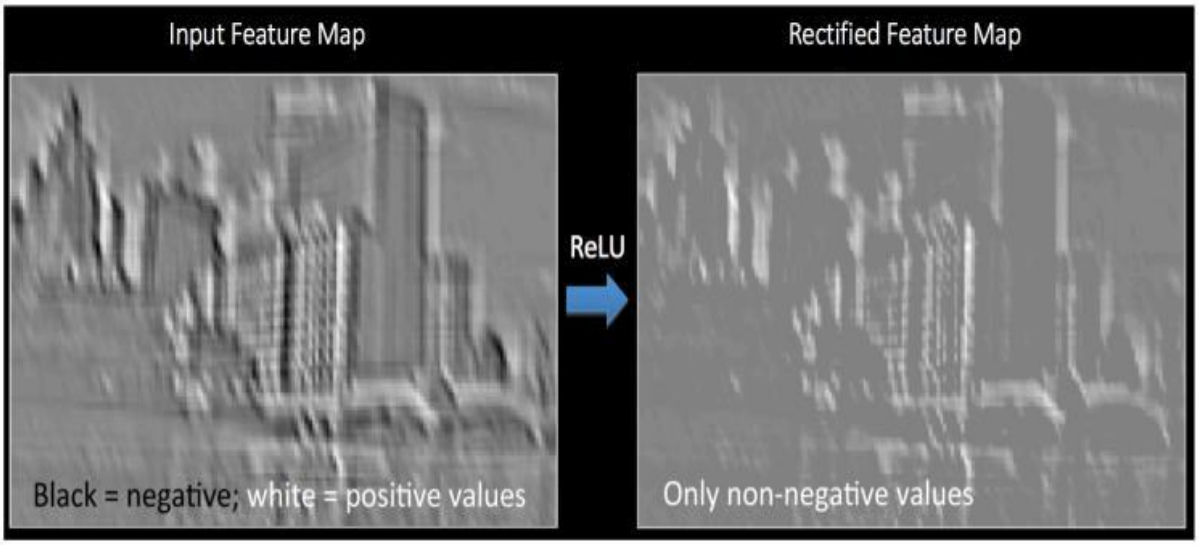


Figure 13: ReLU operation [20]

4.3.3 Pooling

Pooling is a way to take large images and reduce the number of parameters and computation in the network while preserving the most important information in them. It can be of three types

- Max Pooling
- Sum Pooling
- Average Pooling

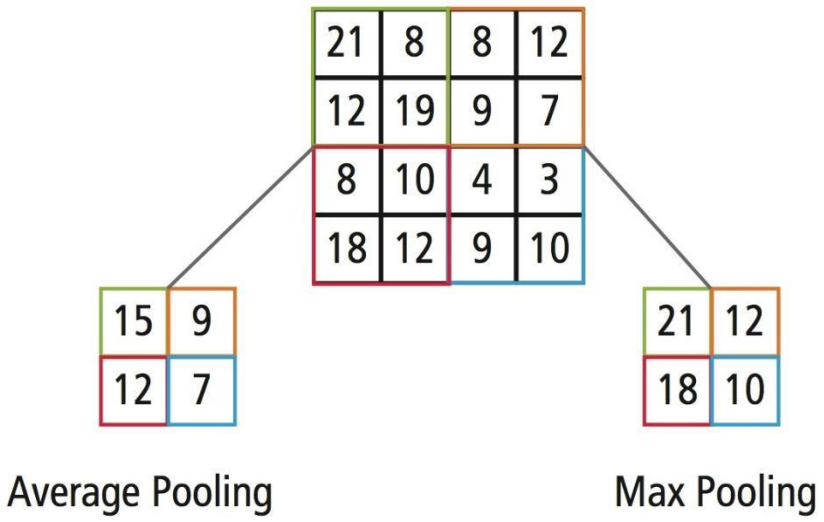


Figure 14: Example of Pooling operation [20]

In Max Pooling, we define a spatial neighborhood and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or the sum of all elements in that window (Sum Pooling).

The function of Pooling is to:

- make the input representations (feature dimension) smaller and more manageable
- reduce the number of parameters and computations in the network, therefore, controlling overfitting [20]
- make the network invariant to small transformations, distortions, and translations in the input image
- arrive at an almost scale-invariant representation of the image

4.3.4 Fully Connected Layer

The term "Fully Connected" means that every neuron in the previous layer is connected to every neuron in the next layer. It is a traditional Multi-Layer Perceptron that uses a softmax activation function in the output layer. Other classifiers like SVM can also be used. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

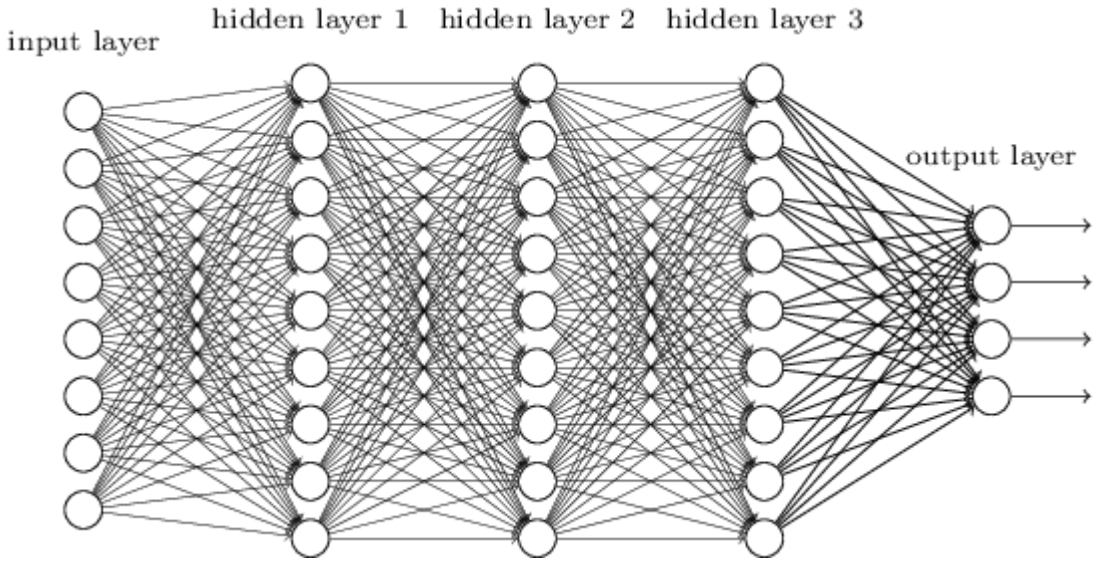


Figure 15: Fully connected layer [20]

4.4 Hyperparameters

Hyperparameters are parameters that cannot be learned directly from the regular training process. They are fixed before the actual training process begins.

Depth: It corresponds to the number of filters we use for the convolution operation

Stride: It is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time. Having a larger stride will produce smaller feature maps.

Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. It allows us to control the size of the feature maps. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution.

4.5 Pre-trained models

Pre-trained model is a Convolutional Neural Network which has been trained on a relatively large-scale problem such as ImageNet, can be used in other visual recognition tasks without the need to train the first few layers. Those fixed layers are fixed feature detectors. The early layers of any ConvNet will learn almost the same such "basis" features such as edge, color or gradient features for visual recognition tasks. Only the higher-level features tend to be class specific that is why when using a pre-trained network, it is important to re-train some of the upper layers.

4.5.1 VGG16/VGG19

With the rise in the use of ConvNet in Image Recognition and Classification, a number of attempts are made to improve the architecture proposed by Krizhevsky et al. (2012) [21] to achieve better accuracy and lower error rate. Some improvements are smaller receptive window size and smaller stride of the first convolutional layer [22], [23] and training and testing the networks densely over the whole image and over multiple scales [23].

VGG addresses another important aspect of ConvNet architecture design – its depth. [24] It fixes other parameters of the architecture, and steadily increase the depth of the network by

adding more convolutional layers, which is feasible due to the use of very small (3 × 3) convolution filters in all layers. To measure the improvement brought by the increased ConvNet depth in a fair setting, all its ConvNet layer configurations are designed using the same principles, inspired by Ciresan et al. (2011) [25]; Krizhevsky et al. (2012). [21]

4.5.2 ResNet50

ResNet is a residual learning framework to ease the training of networks that are substantially deeper than those used previously. It reformulates the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. These residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset, it is evaluated with a depth of up to 152 layers—8 × deeper than VGG nets but still having lower complexity.

When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. [26] Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. [26] So ResNet addresses the degradation problem by introducing a deep residual learning framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, ResNet explicitly let these layers fit a residual mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(x)$, it let the stacked nonlinear layers fit another mapping of $\mathcal{F}(x) := \mathcal{H}(x) - x$. The original mapping is recast into $\mathcal{H}(x) + x$. (As shown in figure 16)

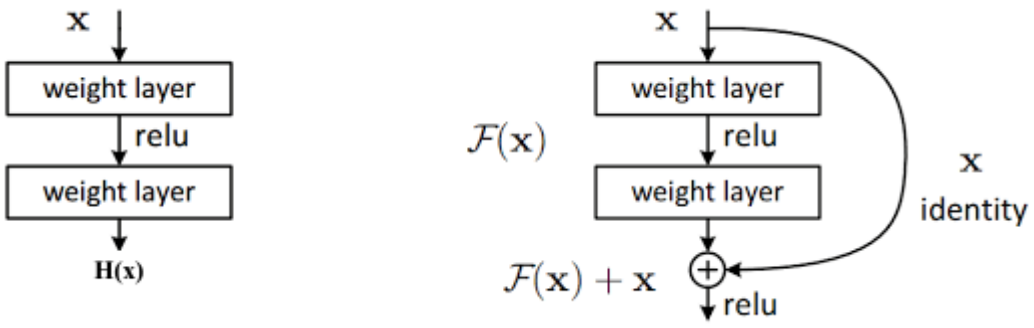


Figure 16: Shallow learning & Residual learning: a building block. [26]

The formulation of 𝓕(𝒙) + 𝒙 can be realised by feedforward neural networks with "shortcut connections" (Fig. 16). Shortcut connections [27] are those skipping one or more layers. In ResNet, the shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers (Fig. 16). Identity shortcut connections add neither extra parameter nor computational complexity. The entire network can still be trained end-to-end by SGD with back-propagation.

4.5.3 InceptionV3

The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. The benefits of the architecture are experimentally verified on the ILSVRC 2014 classification and detection challenges, where it significantly outperforms the current state of the art. [28]

Figure 17: Comparison of VGG16, ResNet-50 and VGG16 [30]

InceptionV3, ResNet-50, ResNet-152, and VGG16 were tested using the ImageNet data set. Tests were run on Google Compute Engine, Amazon Elastic Compute Cloud (Amazon EC2), and an NVIDIA® DGX-1™. Figure 17 shows the results of the same. Most of the tests were run with both synthetic and real data.

CHAPTER 5: IMPLEMENTATION

5.1 Hardware Specifications

The hardware specification and mounting of the camera:

- Minimum 640x480 pixels resolution
- The camera should be positioned on the dashboard such that only the driver's upper body including hands is clearly visible
- The camera should be stationary

The machine on which we implemented Viola-Jones Algorithm has following specifications: 8 GB of RAM, Intel i3 processor clocked at 1.9GHz, 2GB of GPU.

5.2 Software Specifications

We are using MATLAB 2016a software for implementation using computer vision toolbox provided by Mathworks.

5.3 Implementation of Face Detection

5.3.1 Data Retrieval

A sample YouTube video consisting of a person driving a car, such that all specifications mentioned above were met.

5.3.2 Basic Flow of The Face Detection Application

The flowchart shows working of face detection model in Fig. 18. We are processing the video frame in MATLAB. We have used Viola-Jones algorithm to detect a face in the given video frame, and it uses Haar features.

We created face detector object and used the front face profile to detect the face. It uses the pretrained model. The module is sub-divided into two parts, which are: detection of the face, tracking of the detected face. As, once we detect the face, then it will be efficient and

advisable to track the detected face. For this, we use number points that are found according to facial features in the detected face. To track the detected face, in each frame we look for those set of a number of facial features. If this is greater than the defined threshold value, we can detect the face just by tracking the detected face.

While tracking, if the number of points according to facial features goes below the threshold value then the execution command will be given to detection from the tracking and new numPts will be calculated and thus, this process will be continued.

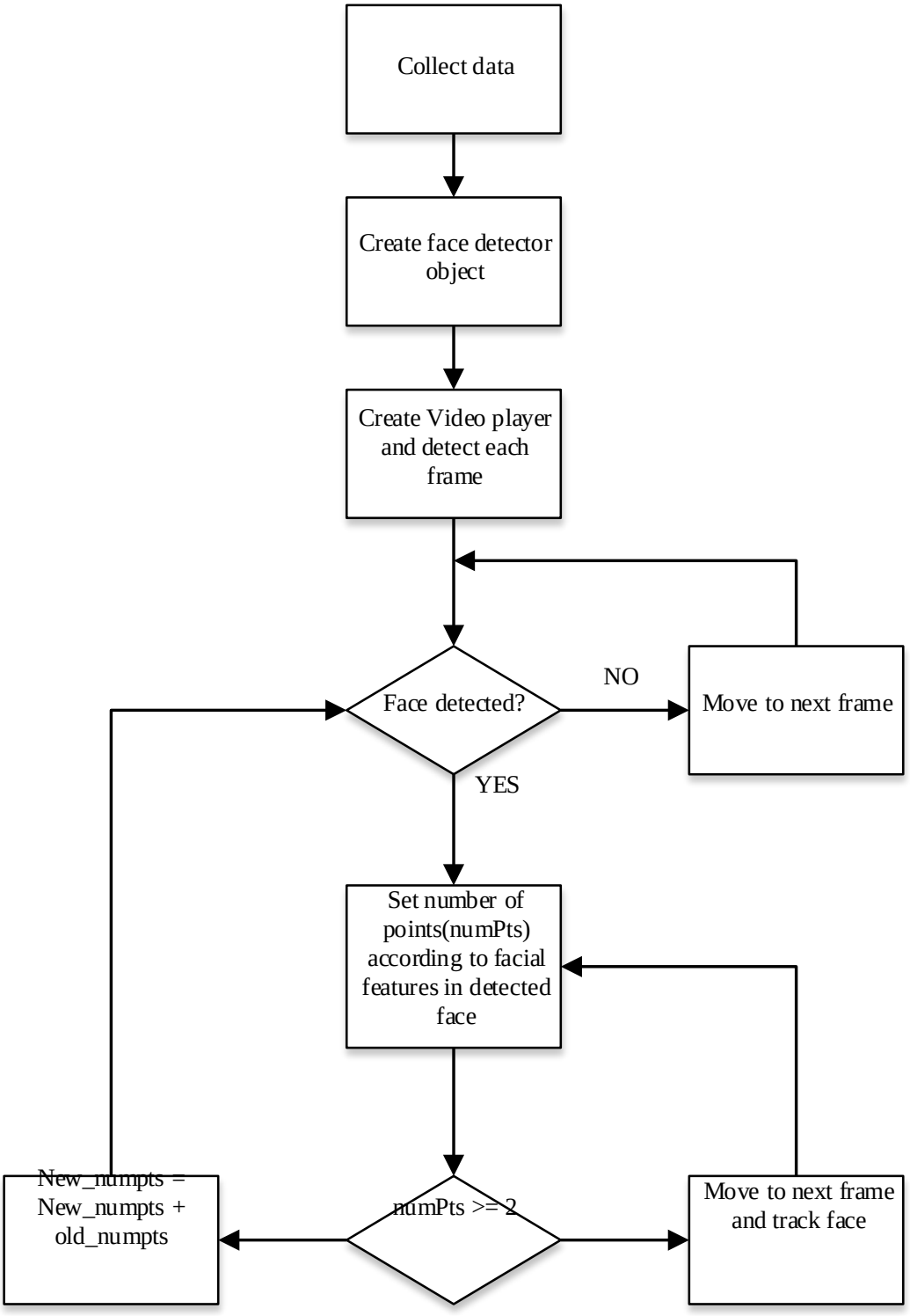


Figure 18: Flow of face detection

5.3.3 Results of Face Detection Model

We were successfully able to test the Viola-Jones algorithm on the data and we achieved following results.

Following are some images of the output of the algorithm implemented:

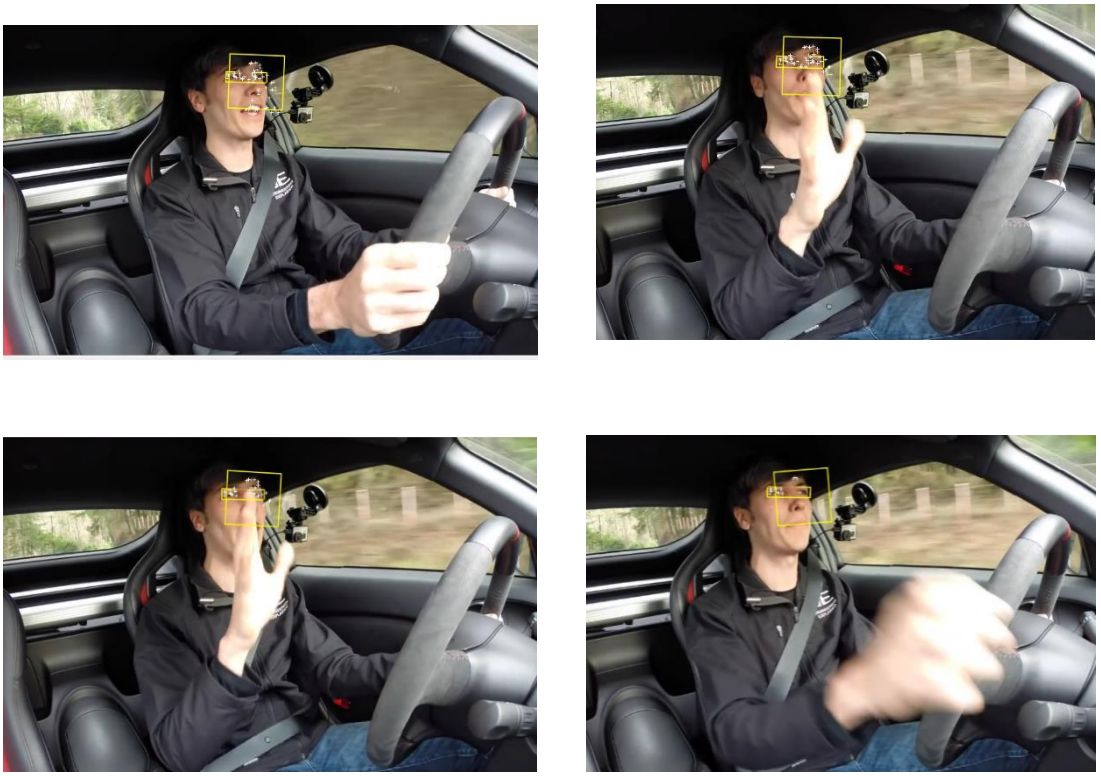


Figure 19: Output of the algorithm

5.3.4 Shortcomings of Model Used for Face Detection

The followings are the shortcomings of the proposed algorithm:

- We might get multiple detections of the same face, due to overlapping sub-windows.
- Sensitive to lighting conditions and does not work properly at night.
- Does not give desired output when used with moving camera

The above method to perform facial recognition perfectly illustrates the features of the algorithms used to detect facial features of the driver.

The above results clearly depict the effectiveness of the applied algorithm in detecting a face as well as a pair of eyes. We can thus, successfully use this algorithm for further application in the project.

5.4 Implementation of Analogous Surveillance Application

We have implemented a surveillance application that is analogous to the objectives of our project. This application sends a notification to the user if any person is detected in the frame area of the room.

5.4.1 Data Retrieval

We are using live video stream through laptop webcam.

5.4.2 Flow of Sending Notification

The system tries to detect a human body in the input video stream. Whenever the human body is detected in the video stream, we alert the user by sending notification on his/her Android Mobile Application. It also captures the snapshot of the current frame in which the human body was detected, and send it to the user as a notification.

We are using Firebase Cloud Messaging to send Notification. It works as described below:

When the human body is detected in the frame, the API request is made to sending the notification to the android device of the user. The snapshot of the current frame is uploaded to Dropbox and the URL of the same is sent to the API which is hosted on Heroku server which has 512 MB of RAM, 1 CPU share and does not have dedicated server. This API requests the Firebase Cloud Messaging to send the cloud message to the given client's Android Mobile Application. After getting the cloud message response from Firebase Cloud Messaging, the android mobile application creates the notification as well as display the photo of the driver. Fig. 20 depicts the flow of the sending notification to the user's device.

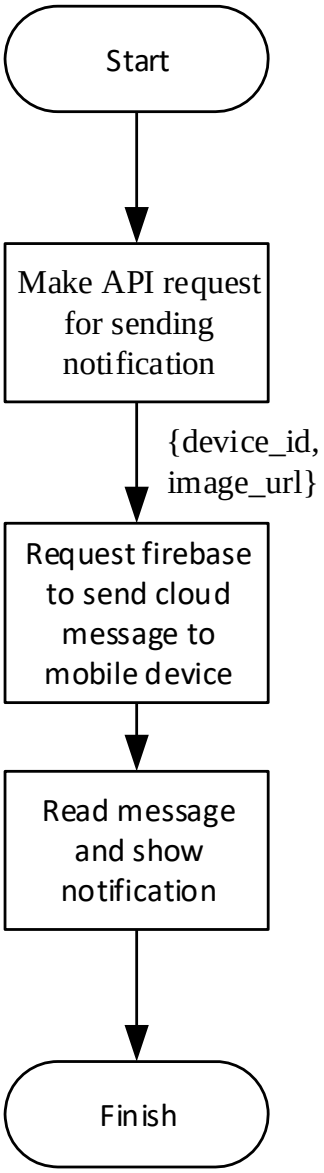


Figure 20: Flow of the notification

5.4.3 Firebase

Firebase is a mobile and web application platform with tools and infrastructure designed to help developers build high-quality apps. Firebase is made up of complementary features that developers can mix-and-match to fit their needs. Firebase Cloud messaging is one of the utility tool provided by Firebase. Firebase Cloud Messaging (commonly referred to as FCM), formerly known as Google Cloud Messaging (GCM), is a cross-platform solution for messages and notifications for Android, iOS, and Web applications.

An FCM implementation includes an app server that interacts with FCM via HTTP or XMPP protocol and a client app. You can compose and send messages using the app server or the Notifications console.

Firebase Notifications is built on Firebase Cloud Messaging and shares the same FCM SDK for client development. For testing or for sending marketing or engagement messages with powerful built-in targeting and analytics, you can use Notifications. For deployments with more complex messaging requirements, FCM is the right choice.

The server side of Firebase Cloud Messaging consists of two components:

- FCM connection servers provided by Google. These servers take messages from an app server and send them to a client app running on a device. Google provides connection servers for HTTP and XMPP.
- An app server required to be built which is hosted on Heroku. The app server sends data to a client app via the chosen FCM connection server, using the HTTP protocol.

A full FCM implementation requires both a client implementation and a server implementation.

5.4.4 API Description

The API consists of two end points. One is used for adding user's unique token to the database which is used to send the notification to a particular device. Another API is used to send notification through Firebase which takes image_url and device's unique id in JSON input data in POST request. This API then sends a request to FCM connection server to send notification on user's device.

5.4.5 Shortcomings

The followings are the shortcomings of the application:

- It requires the reliable data connection.
- Static camera position
- Proper illumination is required for detection

5.5 Proposed model for Distracted Driver Activity Detection

Fig. 21 describes the whole model of the DDAD. The model is sub-divided into two parts: Detection using Image Processing and Classification using Machine Learning.

First of all, one frame is extracted from the video source at some fixed interval. After extracting the frame from the video source, the very first task is to detect a face. If the face of the driver is not detected, then that means either driver is gazing around or talking to someone behind the front seat. Clearly, the driver is distracted in those cases. So, execution will be moved to the classification part to classify the type of the distraction of the driver.

Now, if the profile face of the driver is detected that means the side face of the driver is distracted, then the driver will be considered as distracted either he/she is talking to someone on the front seat or looking outside. The type of the distraction will be known from the result of the classification and driver will be alerted about it.

If the driver passes the face detection test, then we further check for the hand activities of the driver. For the position of the hands, we have created three 3 cases:

1. Both hands on steering (Not distracted)
2. One hand on steering and other on gear stick (Not distracted)
3. Otherwise (Classify the type, High chance of distraction)

If either of the first two cases, then the driver will not be considered as distracted and execution will be moved to the detection of the next frame. For the 3rd case, first the image will be classified and the type of the distraction will be sent to the driver to alert him/her to focus on driving rather than doing other things. The type of distraction may include: drinking, eating, doing makeup, talking on the phone, texting, or operating the radio while driving, as all these activities involve at least one hand to finish the activity.

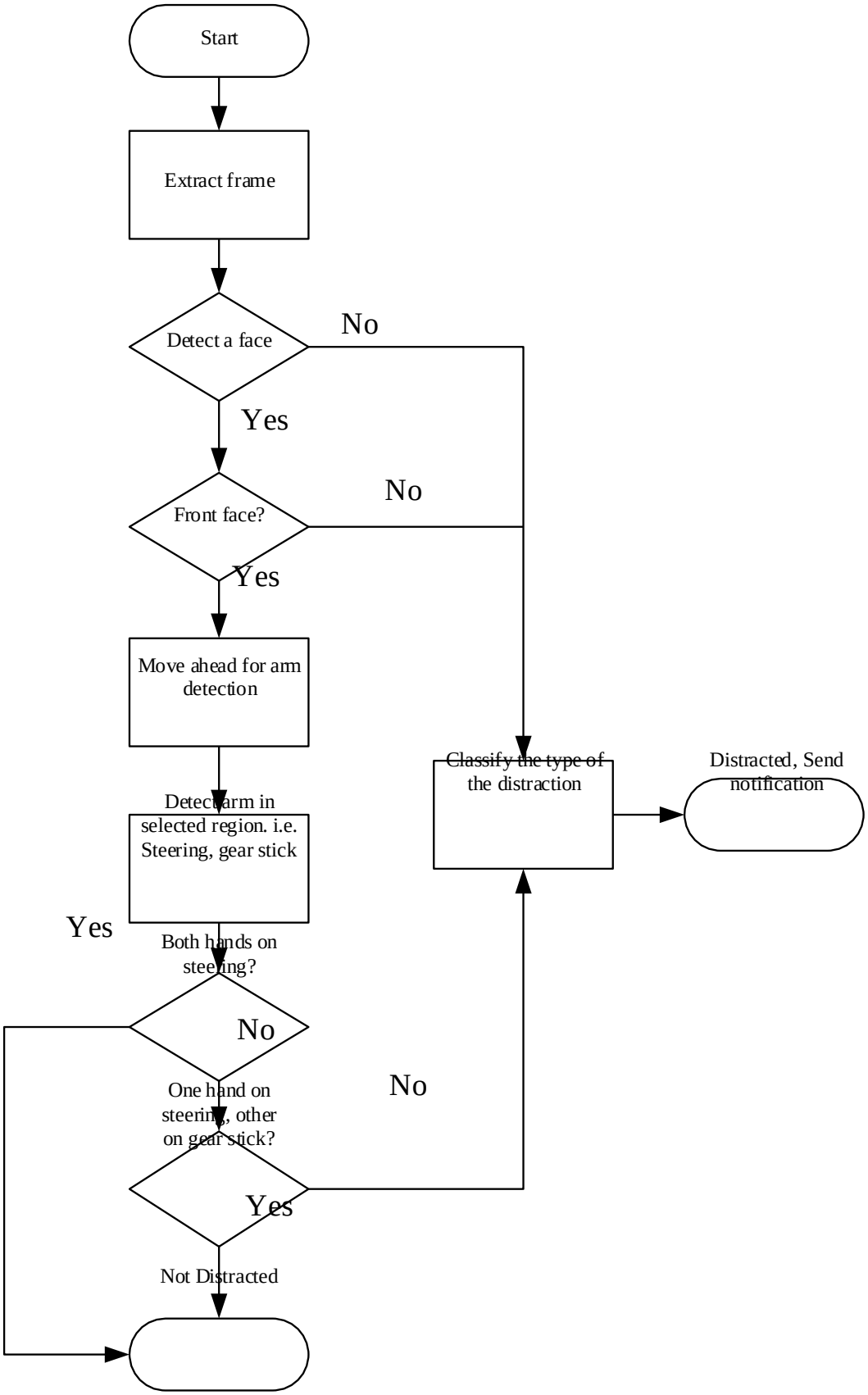


Figure 21: Flow of the application

5.6 Classification model

A VGG-like classification model was built using convolutional neural network in python which was trained and tested on a dataset.

5.6.1 Dataset

We used the training and testing dataset provided by Kaggle Datasets. The training dataset is categorized into 10 classes as follows:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: looking around

The train and test data are split on the drivers, such that one driver can only appear on either train or test set.

5.6.2 Hardware Requirements

The model was trained and tested on the machine with the following specification:

- GTX 980M 8 GB
- 32 GB RAM
- Intel i7 2.7 GHz

5.6.3 Software Requirements

We used following python libraries:

- TensorFlow 1.0.0

TensorFlow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use.

- NumPy

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of level mathematical functions to operate on these arrays.

- OpenCV

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision.

- Keras

Keras is an open source neural network library written in Python. It is designed to enable fast experimentation with deep neural networks, it focuses on being minimal, modular and extensible.

- Pickle

Pickle is the standard mechanism for object serialization. It uses a simple stack-based virtual machine which records the instructions used to reconstruct the object. Pickle, here is used for managing (saving/ loading) cached data.

- CUDA toolkit 8.0

CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows to use CUDA-enabled GPU for general purpose processing.

- CuDNN library by Nvidia

The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

5.6.4 Model architecture

We started with very basic model with two convolutional layers of 64 filters, and FC layers. Then, we expand our model which has similar architecture to VGG16. We replace the topmost fully connected layer of VGG16 with final softmax output layer since we need to classify images in 10 classes.

5.6.5 Training

Before training the model, the training dataset is divided into training data and validation data using k-fold. K-fold divides the data into k groups of samples, of which k – 1 will be used for training the model while the left-out fold will be used as a validation data to validate the model.

Epoch:

A full pass over all of your training data. One should choose the value of epoch wisely. Smaller value of epoch may result into under fitted model, while much higher value of epoch cause the model to be over fitted.

Loss:

A scalar value that we attempt to minimize during the training of the model. The lower the loss, the closer our predictions are to the true categories. Loss is usually Mean Squared Error, or often in Keras, Categorical Cross Entropy. It is Categorical cross entropy between an output tensor and a target tensor, where the target is a tensor of the same shape as the output.

Learning rate

It highly effects the accuracy of the model. One should have high training loss in the beginning with low validation accuracy, but it should decrease very quickly, otherwise it is

pretty much pointless to keep running the model. We tried different learning rate ranging from 0.1 to 0.0001 of which 0.001 worked pretty well.

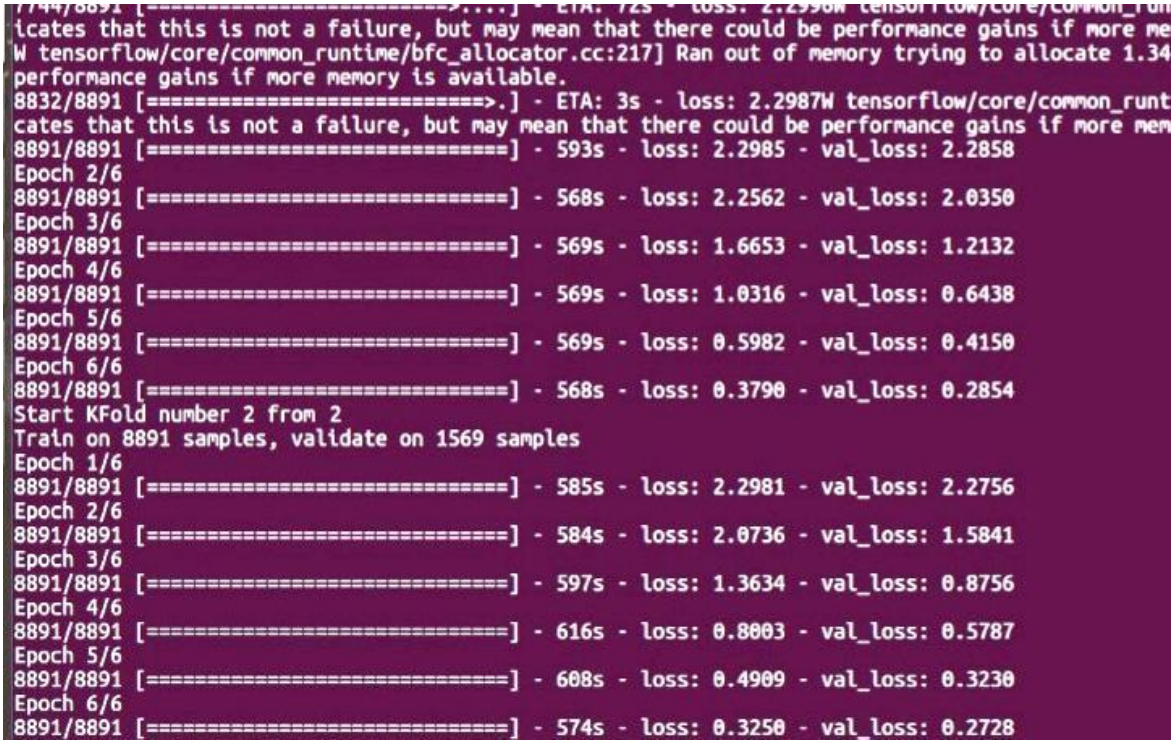


Figure 22: Training of VGG-like model with 2 folds and 6 epochs

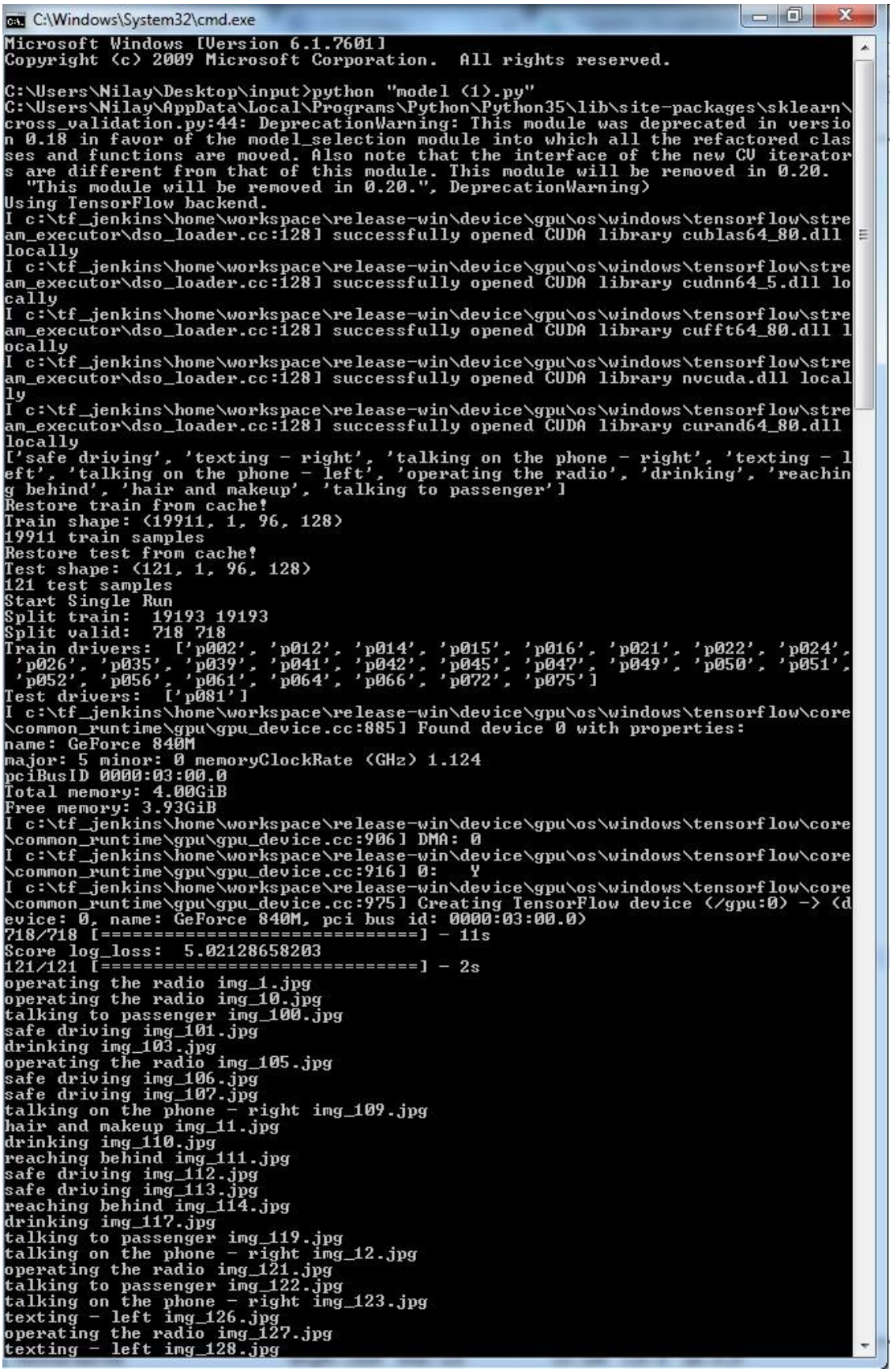


Figure 23: Reading trained model from cache

5.6.6 Testing and Results

The trained model was tested on dataset provided by Kaggle. Following figures depicts the result of the training. The model with only two convolutional layers with 64 filters results into the logarithmic loss of around 5. While the VGG-like model gains logarithmic loss of 0.5. Log loss quantifies the accuracy of a classifier by penalising false classifications. Minimizing the Log Loss is basically to maximizing the accuracy of the classifier.

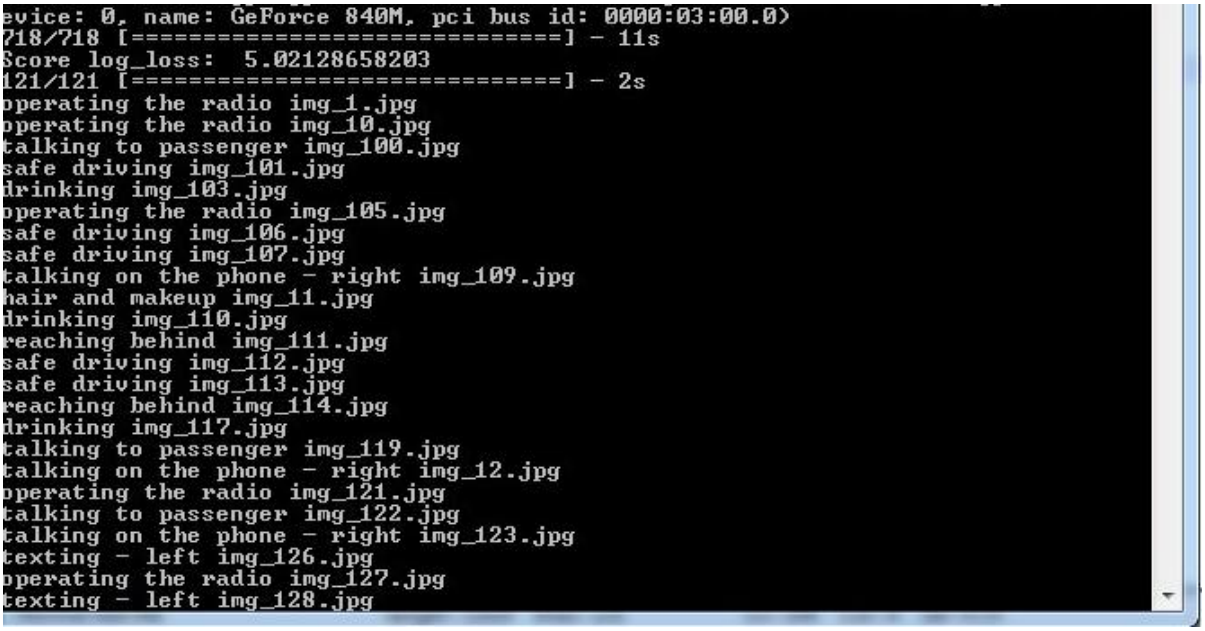


Figure 24: Testing results of Simple 2 Layered model

```
Start testing.............
Score log_loss: 0.50245464545
5000/5000 [==============================] - 356s
operating the radio img_1.jpg
operating the radio img_10.jpg
safe driving img_100.jpg
safe driving img_101.jpg
drinking img_103.jpg
operating the radio img_105.jpg
safe driving img_106.jpg
safe driving img_107.jpg
talking on the phone - right img_109.jpg
texting - right img_11.jpg
drinking img_110.jpg
safe driving img_111.jpg
safe driving img_112.jpg
reaching behind img_113.jpg
reaching behind img_114.jpg
talking on the phone - right img_117.jpg
talking on the phone - left img_119.jpg
texting - right img_12.jpg
operating the radio img_121.jpg
texting - left img_122.jpg
drinking img_123.jpg
texting - left img_126.jpg
safe driving img_127.jpg
texting - left img_128.jpg
texting - left img_129.jpg
talking on the phone - right img_13.jpg
texting - right img_130.jpg
reaching behind img_131.jpg
safe driving img_132.jpg
texting - right img_134.jpg
reaching behind img_135.jpg
talking on the phone - left img_136.jpg
safe driving img_137.jpg
safe driving img_140.jpg
drinking img_141.jpg
operating the radio img_142.jpg
safe driving img_143.jpg
talking on the phone - left img_145.jpg
```

Figure 25: Testing results of VGG-like model

5.7 Implementation of real time Distracted Driver Detection

This section describes how the work flow of the final application including both android app and API. The android app is used to capture an image of a driver and then is transmitted to API for testing. API classifies the image and sends result back to as API response. The driver is alerted by an audio alert if found distracted.

5.7.1 Constraints

Network speed and reliability: Requires reliable and high speed network (At least 5 Mbps upload speed)

Day light: The system does not give accurate results in low day light.

Image resolutions: 1280x960 pixels. Higher image resolutions can be used when high speed network (>5 Mbps) is available.

Data compression: Data compression technique is used to pass the image data over the network for faster transfer. Lower image resolutions can result in inaccurate distraction classification. Because of data compression results are compromised to some extent.

Camera position: The camera position is fixed which is not meant to be changed. Changing in the position of the camera will give inaccurate results.

5.7.2 Software Requirements

The software used during the implementation are:

- **Amazon Web Services (AWS)**

Amazon Web Services (AWS) is a subsidiary of Amazon.com that offers on-demand cloud computing platforms for implementing real time detection. The subsidiary, 'Amazon Elastic Compute Cloud' or 'EC2' was used to rent virtual computers whose processing power was used to train the model to make it more accurate as well as for testing the model on real time data.

- **Android**

An application was developed on Android platform which will be used to capture video on the device camera, breakup the video into frames and send it to AWS for processing. Furthermore, the device is used to send an alert to the driver to warn him/her if he is distracted.

5.7.3 Hardware Specifications

The Amazon EC2 P2.xlarge machine, provided by Amazon web services was used which consisted of

- 1 TESLA k80 GPU
- 4 virtual CPU's
- 61 GiB of RAM

- 12 GiB of Graphics Memory
- A ‘high’ network bandwidth.

5.7.4 Working of Android Application

The application captures the driver’s images using the inbuilt mobile phone camera every T seconds. Initially, a ‘setcounter’ variable is set to zero. This variable is a special case for the ‘Looking around’ class of distraction and is only applicable to this class. The ‘Looking around’ class is a special case because it is common among drivers to look left or right while looking at traffic or taking turns. Although, a constant stream of looking around images to the application is an evident case of distraction and it would then be necessary to alert the driver to pay attention. Thus, to work around this anomalous case the ‘setcounter’ variable increments till a count of three even if the images are distracted but it alerts the driver if the counter exceeds three.

The camera captures the image in 800x600 resolution so as to reduce the size of the file to be transferred to the AWS server. This is done to reduce the latency of file transfer due to shortfalls in data transfer speeds. The image which is sent to AWS is stored at server side in case a need arises for debugging.

The captured image is then passed on to the ‘onPictureTaken’ function which accepts 2 parameters as input, image data in base64 format and direction. The conversion to base64 inflates the image size by 33% but reduces an additional round trip delay for the GET function. This reduces the latency of the transfer thereby compensating for the increase in size. The direction parameter is a parameter which indicates the location of the steering wheel in the car (right hand drive or left hand drive) this makes our application region independent therefore making it more flexible to change.

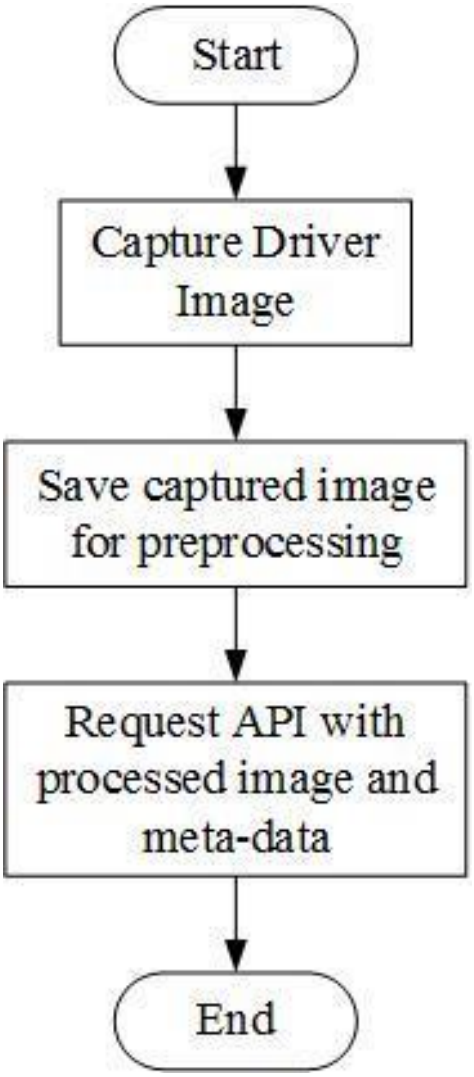


Figure 26: Request Method

The parameters (Encoded ImageData and Direction) are sent to the API as a POST request using RetroFit. RetroFit is a special java library which is described as ‘a type-safe HTTP client for Android and Java’. In essence it turns your HTTP API into a Java interface.

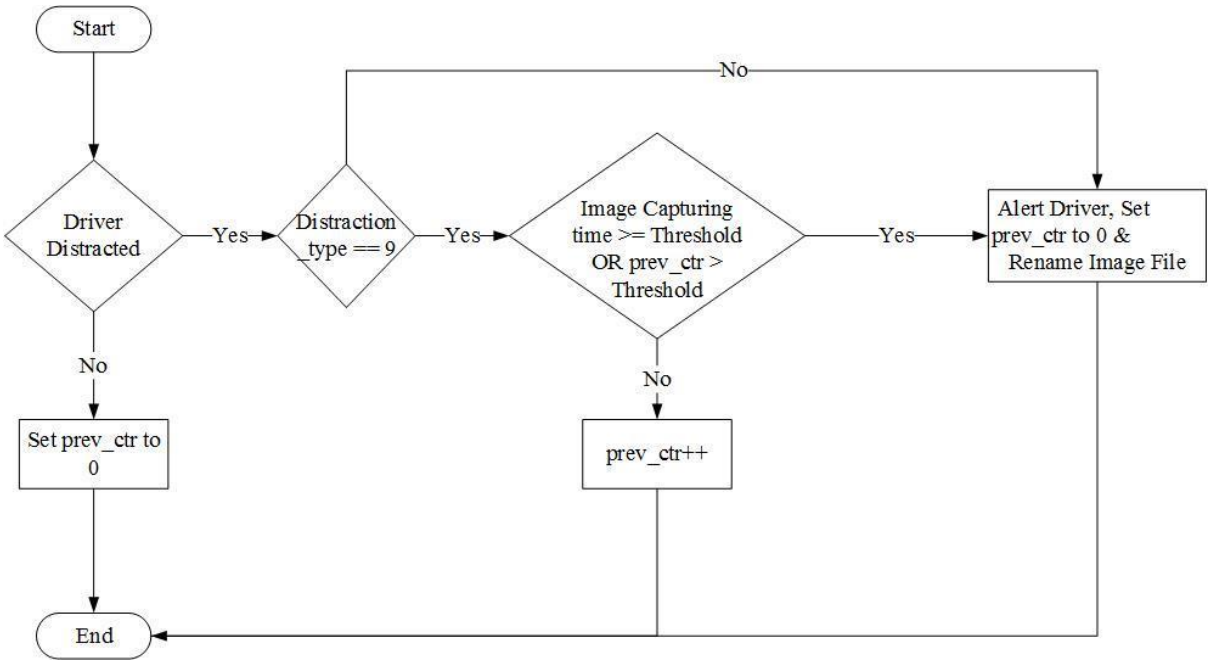


Figure 27: Post Request Methods.

After the AWS server finishes processing it sends its data to the API which then sends it, in JSON format, to the android application. The response by the server consists of two ‘keys’, first being the type/class of distraction and the second being a status message. The type is an integer from 0-9 for the ten classes and returns the error message otherwise.

We considered "looking around" type of distracted as an exceptional case. For example, for turning right or left the car or to change the road lane, the driver has to look in the either mirror, left or right. Therefore, one should not be classified as distracted in such cases. As a solution to this problem, we expanded the scope to declare a driver distracted, i.e. declare a driver distracted only if found distracted in three to four continuous frames taken at regular interval. So that means not to declare a driver distracted if he/she is found looking around in the first frame and then classified as safe driving in the very next frame.

The application, on receiving any type other than ‘0’ (which is the class for safe driving), alerts the driver. First, it sends out an alert sound for a period of 2 seconds if the distraction type is other than looking round and then it sends out a ‘Toast’ (a special type of display format for Android Applications) which displays the type of distraction.

5.7.5 Implementation of API and AWS Server

The model is trained using pre-trained model VGG16. First, model is loaded with 16 layers and then the weights of pre-trained VGG16 model which is trained over half a million images is loaded into the model. Now, the model is trained over the dataset using K-folds method with folds equal to three and the trained model is finally stored in cache to reuse it for classification.

The API along with the model is located on the AWS EC2 machine on the AWS server. On receiving the message from the Android application it forwards it to the testing function of the pre-trained model. The testing phase is clearly depicted in the Figure 28.

The first step is to optimize the classifying process, so by using available resources we can produce the reliable and fast output. The very basic step towards this is to memoization. As, these models are very large in size, loading them into memory is expensive process which takes time. So, it is not feasible to load models at every request and then classify images which will waste time, resources and memory. So, we loaded models into the memory and stored them in one variable that can be used whenever the image is to be classified. Also, repetitive loading and writing of an image are removed.

While training the model, we had introduced K-fold which randomly segments training datasets into k - 1 set, used for training the model and the remaining set is used for testing the efficiency of the model while training it. We selected three folds and in results got three different models trained over different set of images in different order. Therefore, to get a final result, first of all prediction value is calculated using each models. At this point, we have three different predicted value list showing probability of an image to be classified in the particular class. To merge the all three predicted value, arithmetic mean is used. Arithmetic mean of each class is considered as the final result. And whichever class has the maximum probability is considered as classified distraction type.

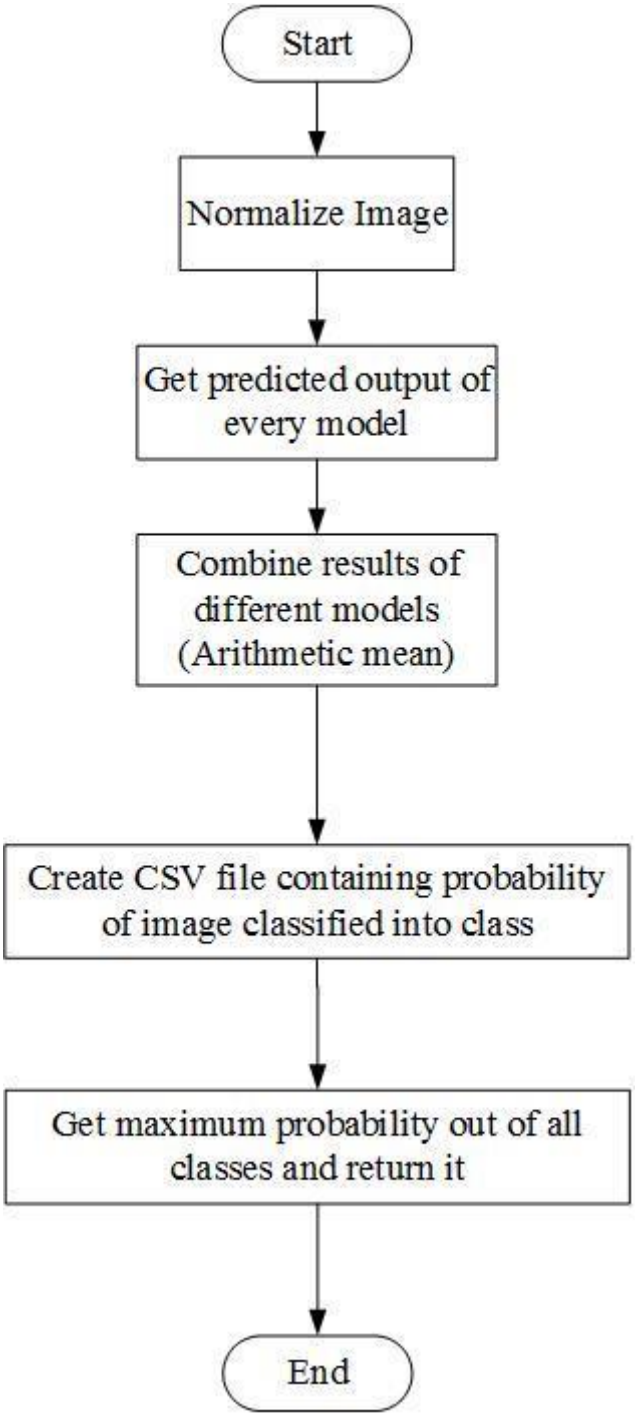


Figure 28: Classification of image.

5.7.6 Results

Results of the systems are shown below.

Figure 29: Example of Safe Driving

Figure 30: Example of Operating the radio

Figure 31: Example of Drinking

Figure 32: Example of Texting

Figure 33: Outliers of Safe Driving

CHAPTER 6: CONCLUSION

With the successful implementation of this project we analysed different image processing techniques and Machine Learning algorithms which would be beneficial in image recognition of a distracted driver. Out of the Machine Learning models we studied, the VGG16 pre-trained model is one of the most suitable machine learning model to implement distracted driver detection. An Android application captures an image at regular interval and sends an API request to classify the image to the AWS server. The application then alerts the driver if found distracted.

REFERENCES

[1] The Indian Express, "Distracted driving causes over 5, 000 deaths each year,". [Online]. Available: http://archive.indianexpress.com/news/distracted-driving-causes-over-5000-deaths-each-year/951382/

[2] C. Craye and F. Karray, “Driver distraction detection and recognition using RGB-D sensor”

[3] "Viola-Jones’ face detection claims 180k features," 2016. [Online]. Available: https://stackoverflow.com/questions/1707620/viola-jones-face-detection-claims-180k-features.

[4] W. B. Albery, D. W. Repperger, G. B. Reid, C. Goodyear, L. E. Ramirez and M. M. Roe, “Effect of noise on a dual task: subjective and objective workload correlates [pilot performance in fighter aircraft cockpit],” IEEE National Aerospace and Electronics Conference: NAECON 1987. Dayton, OH, USA. Vol. 4, pp. 1457- 1463, 18-22 May 1987

[5] J. A. East K. W. Bauer Jr. and J. W. Lanning, “Feature selection for predicting pilot mental workload: A feasibility study,” International Journal of Smart Engineering System Design, Vol. 4, Issue 3, pp. 183-193, Jul./Sep. 2002.

[6] P. Viola and M. Jones, “Rapid Object Detection Using a Boosted Cascade of Simple Features,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, 2001, pp. I–511–I–518 vol.1.

[7] P.Viola and M.Jones, “Robust Real-Time Face Detection, “International Journal of Computer Vision, vol. 57, pp. 137–154, 2004.

[8] L. G. Valiant, “A Theory of the Learnable,” 1984.

[9] Z. Zhang and J. Zhang, “A New Real-Time Eye Tracking for Driver Fatigue Detection,” in the 6th International Conference on ITS Telecommunications Proceedings, June 2006, pp. 8–11.

[10] X. Fan, B.-C. Yin, and Y.-F. Sun, “Yawning Detection for Monitoring Driver Fatigue,” in the International Conference on Machine Learning and Cybernetics, vol. 2, Aug 2007, pp. 664–668.

[11] W. Rongben, G. Lie, T. Bingliang, and J. Lisheng, “Monitoring Mouth Movement for Driver Fatigue or Distraction with One Camera,” in Proceedings of the 7th

International IEEE Conference on Intelligent Transportation Systems, Oct 2004, pp. 314–319.

[12] Y. Qiao, S. Yu, P. Su, and L. Zhang, “Research on an iterative algorithm of ls channel estimation in mimo ofdm systems,” Broadcasting, IEEE Transactions on, vol. 51, no. 1, pp. 149–153, March 2005.

[13] Y. Qiao, S. Shimamoto, H. Wang, W. Chen, and C. Song, “Iterative unequal length search syndrome decoding for product codes,” in Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE, March 2008, pp. 593–598.

[14] Y. Qiao, C. Song, and S. Shimamoto, “Rate 2/3 trellis-coded 8-ary elliptical phase shift keying modulation,” in Wireless Communication Systems, 2007. ISWCS 2007. 4th International Symposium on, Oct 2007, pp. 161–165.

[15] Y. Qiao, K. Zeng, L. Xu and X. Yin, "A Smartphone-Based Driver Fatigue Detection Using Fusion of Multiple Real-Time Facial Features"

[16] P. Paul and M. Gavrilova, “PCA Based Geometric Modeling for Automatic Face Detection”, 2011 International Conference on Computational Science and Its Applications, pp. 33- 38.

[17] M. Chauhan, and M. Sakle, “Study & Analysis of Different Face Detection Techniques”

[18] "Head, eye, and hand patterns for driver activity recognition - IEEE Xplore document," 2016. [Online]. Available: http://ieeexplore.ieee.org/document/6976834/.

[19] "Neural network architectures," 2016. [Online]. Available: http://culurciello.github.io/tech/2016/06/04/nets.html.

[20] "CS231n Convolutional neural networks for visual recognition," [Online]. Available: http://cs231n.github.io/convolutional-networks/.

[21] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural net- works. In NIPS, pp. 1106–1114, 2012.

[22] Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. CoRR, abs/1311.2901, 2013. Published in Proc. ECCV, 2014.

[23] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In Proc. ICLR, 2014.

[24] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[25] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. Flexible, high performance convolutional neural networks for image classification. In IJCAI, pp. 1237–1242, 2011.

[26] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[27] C. M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995.

[28] S. Arora, A. Bhaskara, R. Ge, and T. Ma. Provable bounds for learning some deep representations. CoRR, abs/1310.6343, 2013.

[29] M. Lin, Q. Chen, and S. Yan. Network in network. CoRR, abs/1312.4400, 2013.

[30] Benchmarks | Tensorflow, Available: https://www.tensorflow.org/performance/ benchmarks

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Mr. R. P. Gohil, Associate Professor in Computer Engineering Department, S.V.N.I.T., Surat, for his valuable guidance, useful comments and co-operation with kind and encouraging attitude at all stages of the experimental work for the successful completion of this work.

We are also thankful to S.V.N.I.T., Surat and its staff for providing this opportunity which helped us in gaining sufficient knowledge and to make this Project Report successful.