Drive and Arrive - Distracted Driver Detection

Department of Industrial Engineering and Operations Research at Columbia University
yx2316@columbia.edu, yl3380@columbia.edu, jl4564@columbia.edu

Abstract: **According to Kaggle competition requirement, we aimed to improve these alarming statistics and better insure their customers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviors. Our project established a system that could import camera image, automatically detect driver's behavior and alarm driver after positive detection. By extracting image feature and validating our prediction model, our accuracy of predicting the class of driver's behavior reached 72%. Meanwhile, we developed an interaction friendly interface to apply in real driving environment.**

Keywords: face recognition, image processing, random forest, distracted driver, caffe

I. INTRODUCTION

A. Motivation

We've all been there: a light turns green and the car in front of you doesn't budge. Or, a previously unremarkable vehicle suddenly slows and starts swerving from side-to-side. When you pass the offending driver, what do you expect to see? You certainly aren't surprised when you spot a driver who is texting, seemingly enraptured by social media, or in a lively hand-held conversation on their phone. According to the CDC motor vehicle safety division, one in five car accidents is caused by a distracted driver. Sadly, this translates to 425,000 people injured and 3,000 people killed by distracted driving every year.

B. Objective

Our aim is to improve these alarming statistics and better insure their customers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviors. By analyzing 2D camera image, we not only recognized whether there is driver in a car, but classified each driver's behavior as well. As for our objective, we aimed to develop a system that could take image as input, analyze image feature, classify image content and display our classification result and alarm via website interface.

II. SYSTEM OVERVIEW

A. System Design and Methodology

Drive and Arrive aims to detect driver distractions and give prompt alerts to correct people's inappropriate driving behaviors. To guarantee the performance as well as the efficiency of the system, the following methodologies are adopted:

1. Face recognition

Assume the camera captures the driver as frequently as once per second, the system will have to process a decent amount of photos and classify each of them to a specific behavior label. To avoid this computationally expensive situation, the distraction detection algorithm will be triggered only after the face of the driver is recognized.

2. Image Feature Extraction

To study driver's behaviors, deep features of the photos are needed. Different frameworks and libraries are applied.

3. Feature Decomposition

The deep features of the images are of very high dimensions, which might impede the classification modeling process. Therefore, Principal Component Analysis is incorporated to balance the tradeoff between the time and accuracy of classification.

4. Classification/Distraction Detection

Since multi-label classification is involved, the candidate models chosen for this project are: Neural Network, K-Nearest Neighbors and Random Forest. We applied cross validation for model selection and fine-tuning to decide on model parameters.

5. Interface Construction

The interface allows user to provide photos for detection. Face recognition, feature extraction and distraction detection will be implemented. Customized alert will be displayed to rectify distracted driving.

B. Tools and Dependencies

1. Spark + Python

The face recognition and feature extraction are implemented in Python because of the library dependency. Packages including Caffe and OpenCV are required for the system to function.

To improve the efficiency of the system, Spark is applied to enable fast large-scale data processing. Specifically, command spark-submit is called to run the python scripts. Model training/classification is developed in Python with Spark as well considering the unity of the system.

2. R Shiny

The final web interface is implemented in R with Shiny. The server will call Spark and run the python scripts automatically once the photo captured is ready.

C. Dataset

The data is collected from a Kaggle competition: State Farm Distracted Driver Detection. Two files are provided:

1. imgs.zip, a zipped folder of all (train/test) images. In total, 4.31 GB of images are collected.
2. driver_imgs_list.csv, a list of image ids, subject (driver) ids, and behavior labels.

There are ten different behavior labels:

Label	Description
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	talking to passenger

III. FACE RECOGNITION

To better warn drivers to drive safe, it's essential to detect distracted driving behavior in a real-time time frame. Thus we think we should come up with an idea to identify the situation when we need to start our detecting program. The most important thing is to reduce computing time and save computing resources. That's why we make great efforts to do face recognition at first step - only do we identify the driver's face will we start to detect distracted driving behavior.

What we are using is Haar-cascade detection model. Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image.

$$\text{sum} = \text{I(C)} + \text{I(A)} - \text{I(B)} - \text{I(D)}$$

where A, B, C, D belong to the integral image I, as shown in the figure.

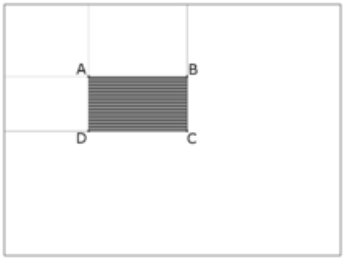

Figure 1 Haas Rectangle

For example, let us say we have an image database with human faces. It is a common observation that among all faces the region of the eyes is darker than the region of the cheeks. Therefore, a common haar feature for face detection is a set of two adjacent rectangles that lie above the eye and the cheek region. The position of these rectangles is defined relative to a detection window that acts like a bounding box to the target object.

The procedures to do face recognition are as follows:

- First, load default face Haar-cascade detection model.
- Identify whether there's a driver's face or not in the picture
- Finally locate the face and use rectangle to point it out

Here are some visualization results of face recognition:

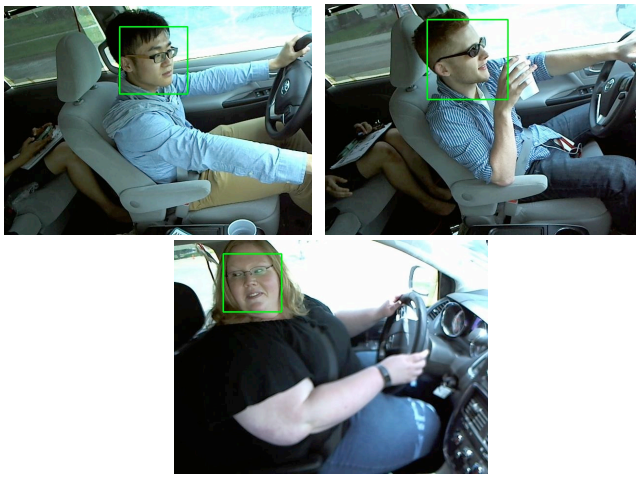

Figure 2 Face Recognition Result

IV. Feature Extraction and Decomposition

A. Feature Extraction

1) Caffe

The Caffe framework from UC Berkeley is designed to let researchers create and explore Convolutional Neural Networks (CNN) and other Deep Neural Networks (DNNs) easily. It provides end-to-end learning to many tasks, especially in image recognition. Particularly, Caffe has a pretrained model, CaffeNet, that is trained on one of the most comprehensive image database, ImageNet, from which the learned features can be extracted from each layer of the neural network.

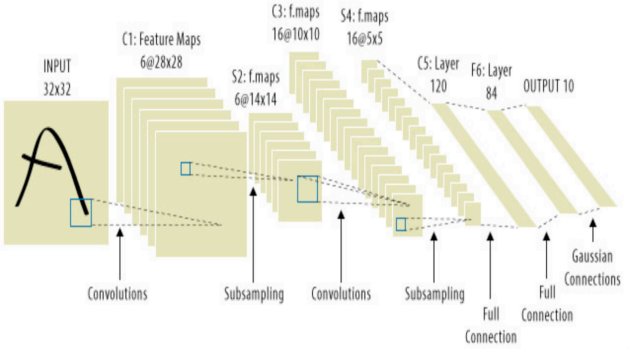

Figure 3 Convolutional Neural Network

The Conv layer implements the following steps[1]:

Accepts a volume of size $W_1 \times H_1 \times D_1$;

Requires four hyper parameters: Number of filters K, their spatial extent F, the stride S, the amount of zero padding P.
Produces a volume of size $W_2 \times H_2 \times D_2$where:

$$W_2 = \frac{(W_1\text{-}F+2P)}{S} + 1, H_2 = \frac{(H_1\text{-}F+2P)}{S} + 1, D_2 = K$$

With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Caffe is recognized as the fastest convent implementation available. It can process over 60M images per day with a single NVIDIA K40 GPU*. That's 1 ms/image for inference and 4 ms/image for learning[2].

We adopted the neural network structure below and chose layer 7/8/9 as candidate features for model training.

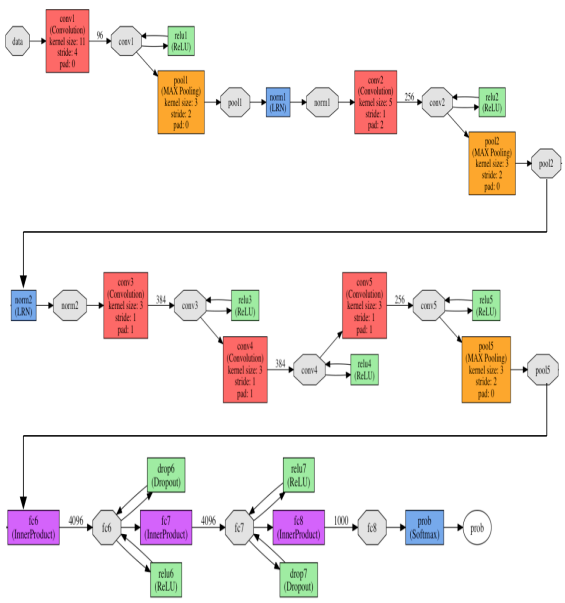

Figure 4 Neural Network of Caffe

Figure 5 Visualization of the layers

2) OpenCV:

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. It was designed for computational efficiency and with a strong focus on real-time applications. In this project we also tried to use OpenCV to extract more features, whose name is SIFT (Scale-invariant Feature Transform).

Scale-invariant feature transform is an algorithm in computer vision to detect and describe local features in images. Similar to some other corner detectors, SIFT is rotation-invariant. What's special about SIFT is that it's also scale-invariant. At first step, SIFT will do a scale-space extrema detection using Difference of Gaussians.

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma)$$

where L(x, y, kσ) is the convolution of the original image I(x, y) with the Gaussian blur at scale kσ.

Once this DoG are found, images are searched for local extrema over scale and space. Once potential key-points locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema.

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

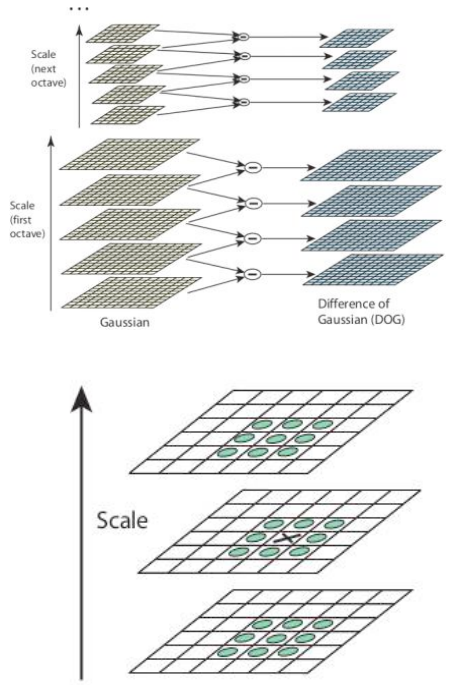

Figure 6 Local Extrema Detection using DoG

Figure 7 SIFT Keypoints Visualization

However, in our case of classification problem, SIFT is not what we are really looking for. It's super helpful for object recognition but not that good for a multi-class classification algorithm.

B. Feature Decomposition

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. This transformation is defined in such a way that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

By doing PCA and finding each principal component, we are actually projecting data in $R^q \; to \; R^P$. Given data points $x_1, x_2, \dots, x_n \in R^p$, we want to reconstruct the data as $f(\lambda) = \mu + v_q\lambda$, where lambda is the low-dimensional data points we are projecting[3]. Creating a good low-dimensional representation of the data requires that we carefully choose $\mu, v_q \; and \; \lambda$. One way we can do this is by minimizing the reconstruction error given by

$$\min_{\mu, \lambda_1 \dots N, v_q} \sum_{n=1}^{N} \left| x_n - \mu - v_q \lambda_n \right|.$$

This is equal to maximizing the variance with the objective function:

$$\min_{v_q} \sum_{n=1}^{N} \left| x_n - \mu - v_q v_q^T x_n \right|^2.$$

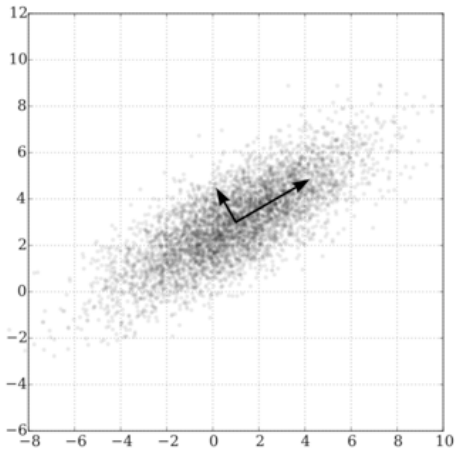

Figure 8 Data and orthogonal transformation

By applying PCA, the dimension of the deep features extracted by CaffeNet and SIFT can be largely reduced without losing much information, which means the majority of the variance in the data can be remained. Refer to Table 1 for detailed PCA performance on Caffe features.

Table 1 PCA performance on Caffe features

Caffe Layer	Original Dimension	Updated Dimension	Variance Remained
7	4096	750	95.41%
8	1000	70	95.77%
9	1000	5	96.15%

As mentioned before, PCA attempts to keep as much variance in the data as possible. However, applying PCA on image features could result in low classification accuracy, since the message of the features will change accordingly, which might impact the classification algorithm. Therefore, there's a trade-off between algorithm efficiency and algorithm accuracy.

V. CLASSIFICATION

A. Classification Algorithm

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

1) Random forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = $x_1, x_2, \ldots, x_n$ with responses Y = $y_1, y_2, \ldots, y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:
Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
Train a decision or regression tree fb on X_b, Y_b..

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f_b}(x')$$

or by taking the majority vote in the case of decision trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample x_i, using only the trees that did not have x_i. in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

2) K nearest neighbor

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

3) Multilayer perceptron neural network

A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the network. MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable.

Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron.

B. Fine-tuning

Since we extracted different layer of caffe result, here we are about to test the influence of each input for our model. Thus we chose random forest classifier, k nearest neighbor classifier and multilayer perceptron classifier to test our input.

Table 2 Input performance

Model / Input	Neural Network	K Nearest Neighbor k = 5	Random Forest n = 10
feature7	54%	69%	72%
feature7_pca	65%	69%	66%
feature8	14%	67%	69%
feature8_pca	56%	67%	67%
feature9	21%	41%	64%
feature9_pca	12%	19%	25%

Feature-I stands for i^{th} layer of caffe result and feature-i_pca stands for i^{th} layer of caffe result after PCA process. Thus, we could learn from Table 2 that we should choose feature7 as our input through the whole modeling process.

C. Cross validation

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. Our goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

According to the limited dataset and concern of biased validation result, we utilize cross validation for random forest and k nearest neighbor algorithm since these 2 algorithms have a better performance in input fine-tuning process.

Using python package sklearn built-in tool, we spilt our dataset into 80% train-set and 20% test set for different parameters of random forest and k nearest neighbor. Here we choose n = 3, 5, 10 to test performance in each parameter. N stands for number of estimator in random forest algorithm and stands for number of class in k nearest neighbor. We got:

Table 3 Cross validation

Algorithm/n	3	5	10
KNN	33%	41%	58%
Random Forest	31%	43%	63%

According to result of cross validation, we found random forest with 10 estimators will have the best performance among all of our objectives. Thus we chose to utilize random forest as our model.

VI. Interface Construction

The user interface is implemented in R Shiny, a package from RStudio that can be used to build interactive web pages and is constructed by server.R and ui.R. The ui file defines the web page which shows the app to the user while the server file acts as a computer that activates the app.

In ui.R, three tabs are created: Introduction, which describes the motivation and the system design; Detection, which allow users to detect a picture of driver; Contact, which contains the contact information of the project authors.

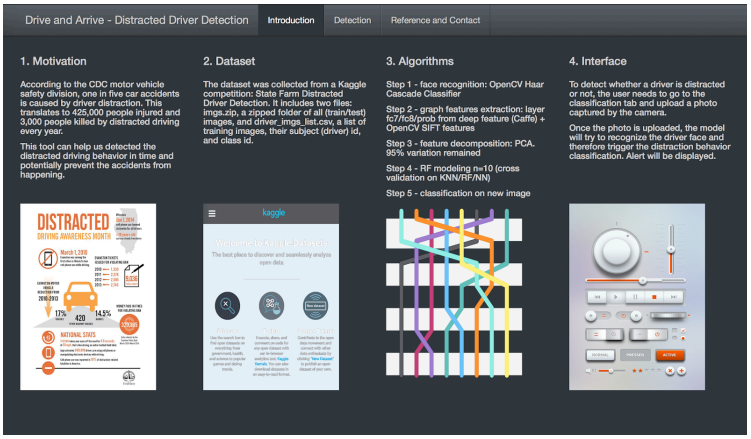

Figure 9 Tab 1 - Introduction

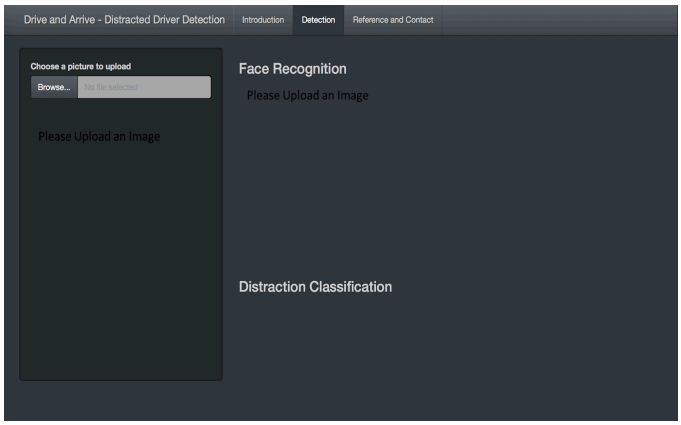

Figure 10 Tab 2 - Detection

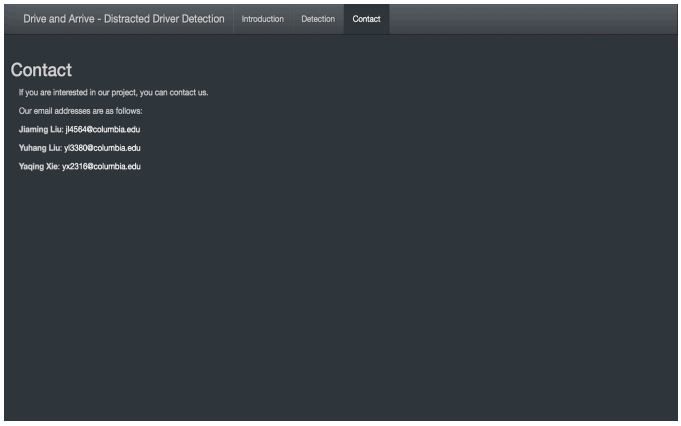

Figure 11 Tab 3 - Contact

In server.R, the code reads the photo uploaded by the user and then triggers Spark to run scripts of face recognition, feature extraction and behavior classification/distraction detection. Figure 9 elaborates the logic behind the interface.

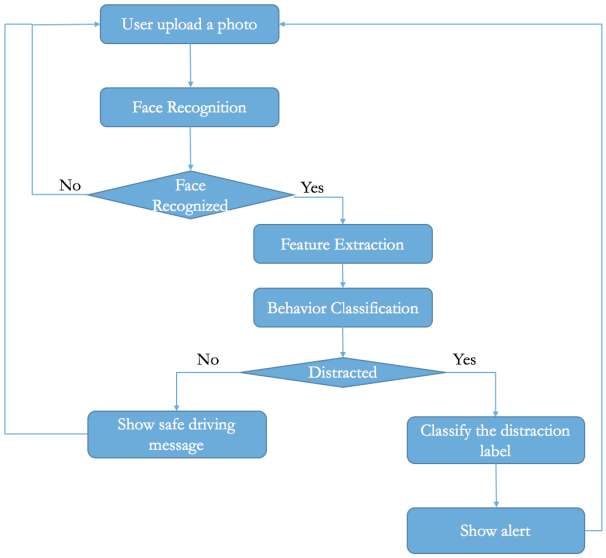

Figure 12 Logic of the server

Once the driver behavior is classified, the system will either show a message to encourage the driver to keep safe driving or give double alerts to alarm the distracted driver. Figure 10 shows the screenshot when the photo uploaded is about a safe driving driver. Figure 11 shows the screenshot when the photo uploaded is about a distracted driver who is operating the radio, where both a pop-up window and a red alert banner are displayed.

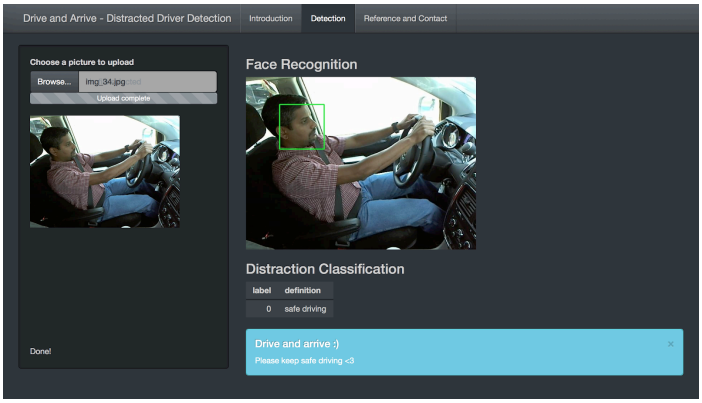

Figure 13 Screenshot 1 - a safe driving driver

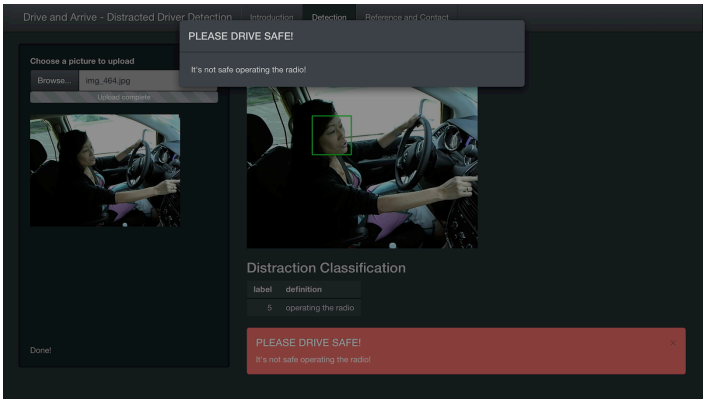

Figure 14 Screenshot 2 - a distracted driver

REFERENCES

[1] http://cs231n.github.io/convolutional-networks/

[2] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014.

[3] http://www.cs.princeton.edu/courses/archive/spr08/cos424/scribe_notes/0424.pdf