Project Synopsis

Title: Detection of drowsiness of driver using Computer Vision.

Performed By: Aditya Srinivasa Rao Ganji

Year: Third Year

College: DJSCE

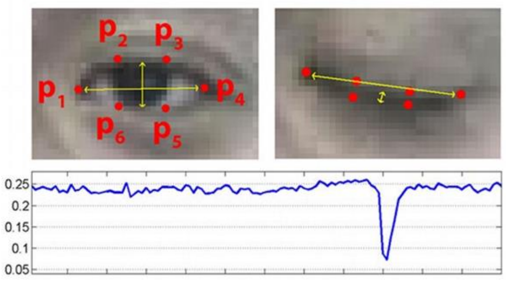
SAP_ID: 60004190007

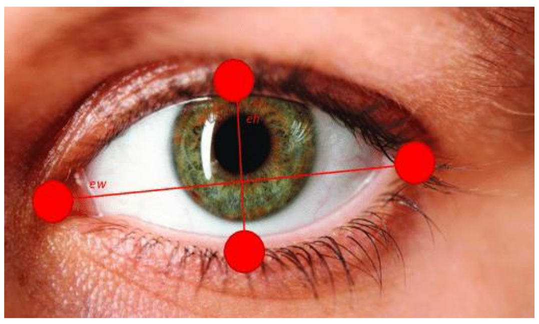
Department: Computer Engineering.

Course: B.Tech in Computer Engineering.

Email: adityaganji889@gmail.com

Phone no: 9082195422,9653124526





Introduction:

Driver fatigue is a significant factor in a large number of vehicle accidents. Recent statistics estimate that annually 1,200 deaths and 76,000 injuries can be attributed to fatigue related crashes. The development of technologies for detecting or preventing drowsiness at the wheel is a major challenge in the field of accident avoidance systems. Because of the hazard that drowsiness presents on the road, methods need to be developed for counteracting its affects.

The aim of this project is to develop a prototype drowsiness detection system. The focus will be placed on designing a system that will accurately monitor the open or closed state of the driver's eyes in real-time. By monitoring the eyes, it is believed that the symptoms of driver fatigue can be detected early enough to avoid a car accident. Detection of fatigue involves the observation of eye movements and blink patterns in a sequence of images of a face.

Dependencies:

The requirement for this Python project is a webcam through which we will capture images. You need to have Python (3.6 version recommended) installed on your system, then using pip, you can install the necessary packages.

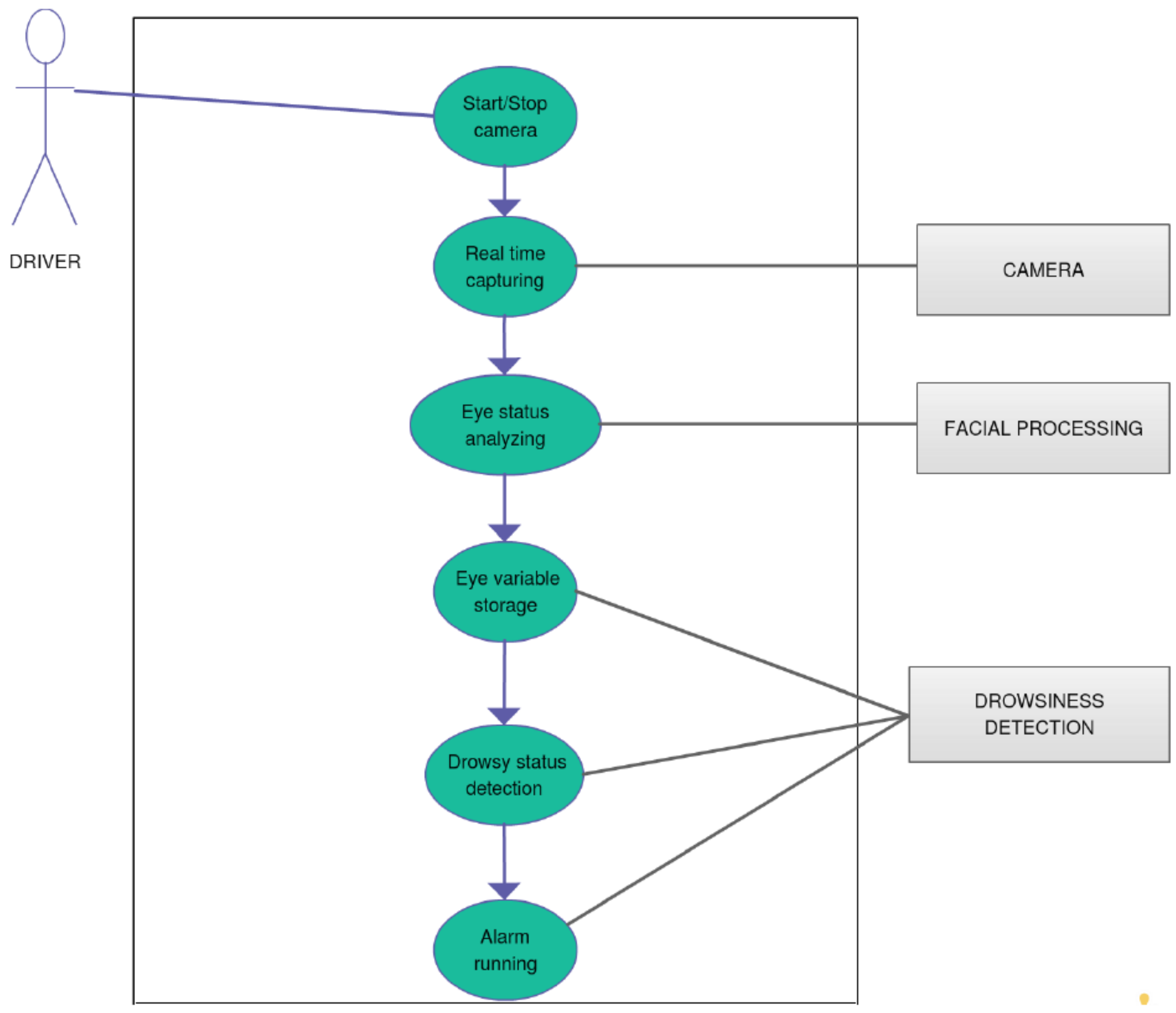
1. **OpenCV –** pip install opencv-python (face and eye detection).
2. **TensorFlow** – pip install tensorflow (keras uses TensorFlow as backend).
3. **Keras** – pip install keras (to build our classification model).
4. **Pygame** – pip install pygame (to play alarm sound).
5. **Matplotlib -** pip install matplotlib (It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK).

Work On:

Method 1:-Using Matplotlib:

First we input the facial image using a webcam. Preprocessing was first performed by binarizing the image. The top and sides of the face were detected to narrow down the area where the eyes exist. Using the sides of the face, the center of the face was found which will be used as a reference when computing the left and right eyes. Moving down from the top of the face, horizontal averages of the face area were calculated. Large changes in the averages were used to define the eye area. There was little change in the horizontal average when the eyes were closed which was used to detect a blink.

Method 2:-Using OpenCv and Keras:



Step 1 – Take Image as Input from a Camera

With a webcam, we will take images as input. So to access the webcam, we made an infinite loop that will capture each frame. We use the method provided by OpenCV, **cv2.VideoCapture(0)** to access the camera and set the capture object (cap). **cap.read()** will read each frame and we store the image in a frame variable.

Step 2 – Detect Face in the Image and Create a Region of Interest (ROI)

To detect the face in the image, we need to first convert the image into grayscale as the OpenCV algorithm for object detection takes gray images in the input. We don't need color information to detect the objects. We will be using haar cascade classifier to detect faces. This line is used to set our classifier **face = cv2.CascadeClassifier(' path to our haar cascade xml file')**. Then we perform the detection using **faces = face.detectMultiScale(gray)**. It returns an array of detections with x,y coordinates, and height, the width of the boundary box of the object. Now we can iterate over the faces and draw boundary boxes for each face.

for (x,y,w,h) **in** faces:

cv2.rectangle(frame, (x,y), (x+w, y+h), (100,100,100), 1)

Step 3 – Detect the eyes from ROI and feed it to the classifier

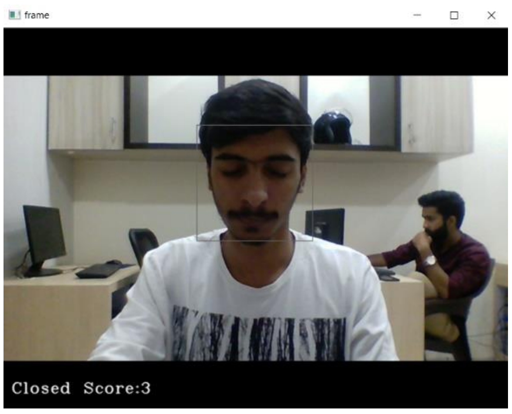
The same procedure to detect faces is used to detect eyes. First, we set the cascade classifier for eyes in **leye** and **reye** respectively then detect the eyes using **left_eye = leye.detectMultiScale(gray)**. Now we need to extract only the eyes data from the full image. This can be achieved by extracting the boundary box of the eye and then we can pull out the eye image from the frame with this code.

l_eye = frame[y : y+h, x : x+w]

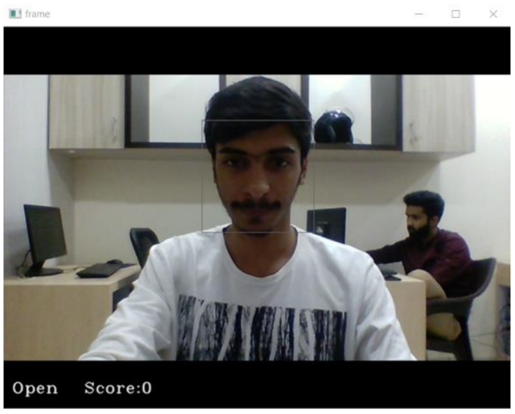
l_eye only contains the image data of the eye. This will be fed into our CNN classifier which will predict if eyes are open or closed. Similarly, we will be extracting the right eye into **r_eye**.

Step 4 – Classifier will Categorize whether Eyes are Open or Closed

We are using CNN classifier for predicting the eye status. To feed our image into the model, we need to perform certain operations because the model needs the correct dimensions to start with. First, we convert the color image into grayscale using **r_eye = cv2.cvtColor(r_eye, cv2.COLOR_BGR2GRAY)**. Then, we resize the image to 24*24 pixels as our model was trained on 24*24 pixel images **cv2.resize(r_eye, (24,24))**. We normalize our data for better convergence **r_eye = r_eye/255** (All values will be between 0-1). Expand the dimensions to feed into our classifier. We loaded our model using **model = load_model('models/cnnCat2.h5')** . Now we predict each eye with our model **lpred = model.predict_classes(l_eye)**. If the value of lpred[0] = 1, it states that eyes are open, if value of lpred[0] = 0 then, it states that eyes are closed.



Close Eye Detection



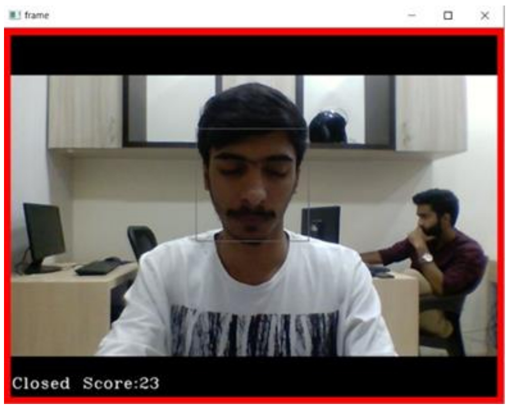
Open Eye Detection

Step 5 – Calculate Score to Check whether Person is Drowsy

The score is basically a value we will use to determine how long the person has closed his eyes. So if both eyes are closed, we will keep on increasing score and when eyes are open, we decrease the score. We are drawing the result on the screen using cv2.putText() function which will display real time status of the person.

cv2.putText(frame, "Open", (10, height-20), font, 1, (255,255,255), 1, cv2.LINE_AA)

A threshold is defined for example if score becomes greater than 15 that means the person's eyes are closed for a long period of time. This is when we beep the alarm using **sound.play()**



Sleep Alert

Work Plan:

Dataset Gathering and Training:

The dataset used for this model is created by us. To create the dataset, we wrote a script that captures eyes from a camera and stores in our local disk. We separated them into their respective labels 'Open' or 'Closed'. The data was manually cleaned by removing the unwanted images which were not necessary for building the model. The data comprises around 7000 images of people's eyes under different lighting conditions. After training the model on our dataset, we have attached the final weights and model architecture file "models/cnnCat2.h5".

- Take image as input from a camera.
- Detect the face in the image and create a Region of Interest (ROI).
- Detect the eyes from ROI and feed it to the classifier.

Classifier will categorize whether eyes are open or closed.

Calculate score to check whether the person is drowsy.

Testing our CNN classifier with eye images in ROI:

The model we used is built with Keras using **Convolutional Neural Networks (CNN)**. A convolutional neural network is a special type of deep neural network which performs extremely well for image classification purposes. A CNN basically consists of an input layer, an output layer and a hidden layer which can have multiple layers. A convolution operation is performed on these layers using a filter that performs 2D matrix multiplication on the layer and filter.

The CNN model architecture consists of the following layers:

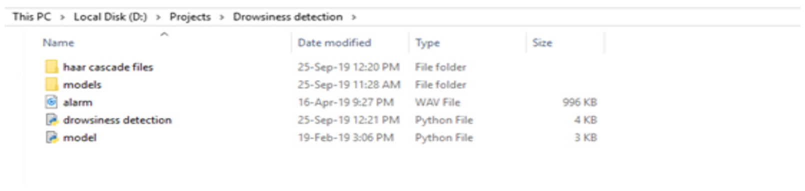
- Convolutional layer; 32 nodes, kernel size 3
- Convolutional layer; 32 nodes, kernel size 3

- Convolutional layer; 64 nodes, kernel size 3
- Fully connected layer; 128 nodes

The final layer is also a fully connected layer with 2 nodes. A Relu activation function is used in all the layers except the output layer in which we used Softmax.

- CNN Classifier will categorize whether eyes are open or closed.
- Then the system will calculate score based on whether eyes are closed or not to check whether the person is drowsy.

Implementation:



This PC > Local Disk (D:) > Projects > Drowsiness detection >

Name	Date modified	Type	Size
haar cascade files	25-Sep-19 12:20 PM	File folder	
models	25-Sep-19 11:28 AM	File folder	
alarm	16-Apr-19 9:27 PM	WAV File	996 KB
drowsiness detection	25-Sep-19 12:21 PM	Python File	4 KB
model	19-Feb-19 3:06 PM	Python File	3 KB

- The “haar cascade files” folder consists of the xml files that are needed to detect objects from the image. In our case, we are detecting the face and eyes of the person.
- The models folder contains our model file “cnnCat2.h5” which was trained on convolutional neural networks.
- We have an audio clip “alarm.wav” which is played when the person is feeling drowsy.
- “Model.py” file contains the program through which we built our classification model by training on our dataset. You could see the implementation of convolutional neural network in this file.
- “Drowsiness detection.py” is the main file of our project. To start the detection procedure, we have to run this file.

Source code:

```
import cv2
import os
from keras.models import load_model
import numpy as np
from pygame import mixer
import time
mixer.init()
sound = mixer.Sound('alarm.wav')
face = cv2.CascadeClassifier('haar cascade files\haarcascade_frontalface_alt.xml')
leye = cv2.CascadeClassifier('haar cascade files\haarcascade_lefteye_2splits.xml')
reye = cv2.CascadeClassifier('haar cascade files\haarcascade_righteye_2splits.xml')
lbl=['Close','Open']
model = load_model('models/cnncat2.h5')
path = os.getcwd()
cap = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
count=0
score=0
thicc=2
rpred=[99]
lpred=[99]
while(True):
ret, frame = cap.read()
height,width = frame.shape[:2]
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25))
left_eye = leye.detectMultiScale(gray)
right_eye = reye.detectMultiScale(gray)
cv2.rectangle(frame, (0,height-50) , (200,height) , (0,0,0) , thickness=cv2.FILLED )
for (x,y,w,h) in faces:
cv2.rectangle(frame, (x,y) , (x+w,y+h) , (100,100,100) , 1 )
```

```
for (x,y,w,h) in right_eye:
r_eye=frame[y:y+h,x:x+w]
count=count+1
r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)
r_eye = cv2.resize(r_eye,(24,24))
r_eye= r_eye/255
r_eye= r_eye.reshape(24,24,-1)
r_eye = np.expand_dims(r_eye,axis=0)
rpred = model.predict_classes(r_eye)
if(rpred[0]==1):
lbl='Open'
if(rpred[0]==0):
lbl='Closed'
break
for (x,y,w,h) in left_eye:
l_eye=frame[y:y+h,x:x+w]
count=count+1
l_eye = cv2.cvtColor(l_eye,cv2.COLOR_BGR2GRAY)
l_eye = cv2.resize(l_eye,(24,24))
l_eye= l_eye/255
l_eye=l_eye.reshape(24,24,-1)
l_eye = np.expand_dims(l_eye,axis=0)
lpred = model.predict_classes(l_eye)
if(lpred[0]==1):
lbl='Open'
if(lpred[0]==0):
lbl='Closed'
break
if(rpred[0]==0 and lpred[0]==0):
score=score+1
cv2.putText(frame,"Closed",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)
# if(rpred[0]==1 or lpred[0]==1):
```

```
else:
score=score-1
cv2.putText(frame,"Open",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)
if(score<0):
score=0
cv2.putText(frame,'Score:'+str(score),(100,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)
if(score>15):
#person is feeling sleepy so we beep the alarm
cv2.imwrite(os.path.join(path,'image.jpg'),frame)
try:
sound.play()
except: # isplaying = False
pass
if(thicc<16):
thicc= thicc+2
else:
thicc=thicc-2
if(thicc<2):
thicc=2
cv2.rectangle(frame,(0,0),(width,height),(0,0,255),thicc)
cv2.imshow('frame',frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
```

Benefits & Limitations:

Benefits:

1. Haar Classifier shows accurate results while detecting face with low false positive rates.
2. Haar Classifier works on low computational cost and a smaller training set, which makes the system economical and the most suitable model for our purpose.

Limitations:

1. Results may not be accurate if training data is less.
2. Results may not be accurate if training images are of different sizes.
3. When the distance between face and webcam is not at optimum range then certain problems are arising.
4. With poor lighting conditions even though face is easily detected, sometimes the system is unable to detect the eyes.
5. If more than one face or face with spectacles is detected by the webcam, then the system gives an erroneous result.

Conclusion:

Successful Drowsiness detection and recognition can be done with OpenCv libraries by training data. The processing capacities required by Matlab were very high. Also there were some problems with speed in real time processing. Matlab was capable of processing only 4-5 frames per second. On a system with a low RAM this was even lower. As we all know an eye blink is a matter of milliseconds. Also a drivers head movements can be pretty fast. Though the Matlab program designed by us detected an eye blink, the performance was found severely wanting. Matlab uses just way too much system resources. With OpenCV, we can get away with as little as 10mb RAM for a real-time application. Although with today's computers, the RAM factor isn't a big thing to be worried about. However, our drowsiness detection system is to be used inside a car in a way that is non-intrusive and small; so a low processing requirement is vital.

Thus we can see how OpenCV is a better choice than Matlab for a real-time drowsiness detection system. We used OpenCV to detect faces and eyes using a haar cascade classifier and then we used a CNN model to predict the status.

References:

[1] http://www.ee.ryerson.ca/~phiscock/thesis/drowsy-detector/drowsy-detector.pdf

[2] http://www.cnblogs.com/skyseraph/archive/2011/02/24/1963765.html

[3] http://www.scribd.com/doc/15491045/Learning-OpenCV-Computer-Vision-with-the-OpenCV-Library

[4]http://opencv.willowgarage.com/documentation/reading_and_writing_images_and_video.html

[5] http://www.scribd.com/doc/46566105/opencv

[6] Learning OpenCV by Gary Bradski and Adrian Kaehler

[7] http://note.sonots.com/SciSoftware/haartraining.html