**Driver Drowsiness Detection System**

**Ayman Mahmud Haque** Computer Science Department Missouri State University

**Abstract**

Drowsiness and Fatigue for drivers are the major cause for road accidents. Detecting the driver eye tiredness is one of the best approaches for detecting drowsiness of a person. An efficient drowsiness detection can prevent a significant number of road accidents and prevent a considerable number of fatalities. As a solution to this problem, we have built a model that will detect whether a person’s eyes are opened or closed and then alert the driver though the alarm sounds.

**Introduction**

Drowsy driving is a common reason for road accidents and various innovative methods are being developed for prevention of accidents due to drowsiness and fatigue. The National Highway Traffic Safety Administration (NHTSA) statistics show that about 2.5% of all fatalities during a crash is attributed to drowsy driving [1]. Drowsiness detection is a safety technology which prevents road accidents that are caused by drivers who fall asleep while driving. In this context, it is important to use new technologies to design and build systems that can monitor drivers and to measure their level of attention during the entire session of driving.

The goal of this project to build a drowsiness detection system that will detect whether a person’s eyes are closed or opened then the system will alert the driver when the drowsiness is detected through some beep sounds. This process is done with the help of Open Source Computer Vision Library (OpenCV) and Convolutional Neural Network (CNN) model. OpenCV is an open source library that helps in building the real time computer vision applications. CNN is a deep neural network that is proven to work efficiently for image recognition and image classification. In this project, face and eyes are the two key parameters to be detected. Haar Feature-based Cascade Classifiers are used to detect the face and eyes from the image. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The dataset used in this project is ‘Closed Eyes in the Wild [2]. This dataset contains 2423 subjects, among which 1192 subjects with both eyes closed are collected directly from Internet, and 1231 subjects with eyes open are selected from the Labeled Face in the Wild (LFW [3]) database. Then the dataset are preprocessed and labelled closed eyes as 1 and open eyes as label 0. The images are then resized to 24X24 pixels.

**Literature review**

Wanzeng et al [4] have used haar features to train their ada boost classifier. To extract haar characteristics and calculation of features the ada boost algorithm is used. Then these trained classifiers are used to detect opened and closed eyes.

Fouzia et al [5] use shape detector algorithm, image processing algorithm to detect eyes and count the eye blink rate followed by drowsiness detection at real time for accuracy without interference.

Shinfeng et al [6] used haar like features and ada boost classifier and active shape model to segment the image into number of rectangle areas, at any position in the original image.

Xiu et al [7] have gone through in depth the influence of components of eye closeness detection, eye patch alignment and feature extraction and classifiers using Local Binary Patterns (LBP), Histograms of Oriented Gradient (HOG), Support Vector Machine (SVM).

Belal et al[8], as the eyes are positioned in the same area, confined their process to area between forehead and mouth. They used Hough transform for circles (HTC) is used to obtain the decision of the eyes state.

In a process proposed by voila et al the face region is detected based on the optimized Jones and Viola method [9]. They use a set of features that are reminiscent of Haar Basis functions.

There are several algorithms and methodologies that have already been proposed and available for face detection which provides the initial idea of detecting the face. However, these techniques are intrusive in nature. This paper [10] is based on Rowley’s eye detection code from the STASM library. This paper [11] observes only the yawning patterns of the rider using two cameras in order to acquire information whether the rider is drowsy or not and this as the reason the dependency on hardware is high.

**Background**

Our drowsiness detection system has been implemented using the python programming language. Below is the list of all the related libraries that we have used to construct our program.

* NumPy
* Os
* Time
* OpenCV2
* Pygame
* Keras.models : Sequential
* Keras. Layers: Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization
* Sklearn

The primary goal of our project was to implement a drowsiness detection system which will determine if a person is drowsy from a video input and alert the driver as soon as it recognizes it. We have implemented a Convolutional Neural Network (CNN) to recognize if eyes are open or closed by training it with a dataset that contains labeled images of open and closed eye-regions. We have implemented the video input and the region of interest extraction through OpenCV and we have used Keras, which is an interface to construct artificial neuron networks, to build and train a CNN model with the dataset to successfully distinguish between Open and closed eyes. With this model, we have implemented the system to recognize drowsiness from the ROI extracted from the video.

In this section, we provide a brief introduction to the tools and libraries mainly used in building our program:

**NumPy**

NumPy is a numerical computing package made for python programming language. It supports Python for calculation of multi-dimensional arrays and matrices and allows high level mathematical functions to be used with arrays. We have extensively used NumPy array to contain and manipulate our image data.

**Sklearn**

Sklearn (formerly known as Scikitlearn) is a robust and dynamic machine learning library that provides a plethora of tools for machine learning and statistical modelling in python. We have used a function from Sklearn called ‘train\_test\_split’ to organize our datset into training and test sets and corresponding labels.

**Matplotlib**

Matplotlib, an extension of NumPy, is a plotting and visualization library that provides an object-oriented API that eases embedding plots into applications and GUIs. We have used matplotlib to visualize our images at certain instances and extensively for visualizing the performance of our CNN.

**OpenCV**

Originally written in C++, OpenCV is an image and video processing library developed by intel, which allows real time computer vision. We have been able take video input from a webcam and to read and manipulate frames from video input and extract ROIs from the frames with the help of different functions from this library. We were able to localize the ROIs with the Haar cascade classifier that comes with this library.

**Keras**

Keras is an open-source library that makes it easy to write deep neural network codes with ease. It is an API that contains blocks of implemented neural network segments such as layers, optimizers, loss functions, activation functions and a wide range of utilities to ease the development of models with the least effort. We have used different modules of Keras to develop our CNN model which we extensively describe in the ‘Methods’ section.

**Methodology**

**Dataset acquisition and preprocessing**

We have searched various repositories for image datasets that aligns with our goal. We were inclined on finding a dataset that contains images of both eyes segregated into two distinguished segments which would essentially save us the nuisance of separating the dataset manually or making a large enough dataset on our own. It was also essential for the dataset to be balanced; hence the distinguished dataset was crucial. It was also essential to find a large enough data set to make our model accustomed to recognizing ROIs more precisely and correctly. We were able to find the “Closed Eyes In The Wild (CEW)” data set that contains facial images of size 100x100 and extracted eye patches of size 24x24 px. In the preprocessing stage, we loaded the images into arrays and classified the images into left and right eyes and in each segment, we divided it into Open and closed labels. The images were inverted and converted into grayscale, resized into 1 channel from 3. After proper labelling and concatenation, the dataset was split into training and testing dataset and labels.

**Description of the main application**

**Algorithm**

**Step 1:** Takes Image as input from web camera

With a webcam, we took video as input. By initializing the webcam, an infinite loop is created that will capture each frame. We used the method provided by OpenCV, cv2.VideoCapture function to access the camera and set the capture object which will read each frame and we store the image in a frame variable

**Step 2:** Detected face from 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. Then we used haar cascade classifier to detect faces and eyes. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. We make use of those XML files. This line is used to set our classifier face = cv2.CascadeClassifier(‘path to our haar cascade xml file’). Then we performed the detection for faces using detectMultiScale() function. If faces are found, it returns the positions of detected faces as Rect(x,y,w,h). Once we get these locations, we can create a ROI for the face and applied eye detection on this ROI.

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

The same procedure is used to detect eyes. First, we set the cascade classifier for eyes in left\_eye and right\_eye respectively then detect the eyes using detectMultiScale(). 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.

**Step 4:** CNN model to classify the eye status

These extracted eyes will be fed into our CNN classifier which will predict if eyes are opened or closed.

**Step 5:** Score is calculated for checking the drowsiness and alerting the driver

We determine how long the person closes his eyes. If the person closes the eyes the score is increased otherwise the score keeps decreasing when the eyes are opened until score is zero. If the eyes are closed for subsequent 15 frames, an alarm is produced to alert the driver. The alarm keeps beeping until the eyes are open and the score is reduced to zero.

**Model development, training and testing**

At the beginning of this section, we want to briefly introduce the modules from Keras used to build the convolutional neural network.

1. Sequential:

This model allows for a plain stack of layers with one input and output tenor.

1. Dense: Densely connected neural network layer, the output of which is activation (dot(input, kernel) + bias)
2. Flatten: Reduces dimension of a matrix or tensor and converts it into one layer.
3. Conv2D: Two-Dimensional Convolutional layer that helps in extracting and learning features from images using image kernels. Filter parameter denotes the number of filters the layer will learn from.
4. MaxPooling2D: Max-pooling down-samples the output from Conv2D layers to reduce the fine detail learning from the feature maps.
5. Dropout: This utility randomly picks out training samples to ignore at each epoch to prevent overfitting
6. Batch Normalization: For increase stability, this normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This reduces loss of information throughout the network.

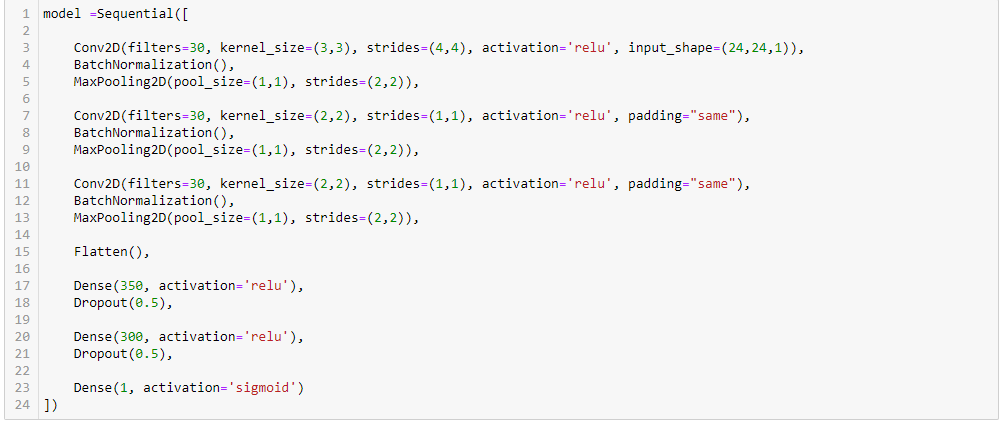


Figure 1: Base CNN model implementation

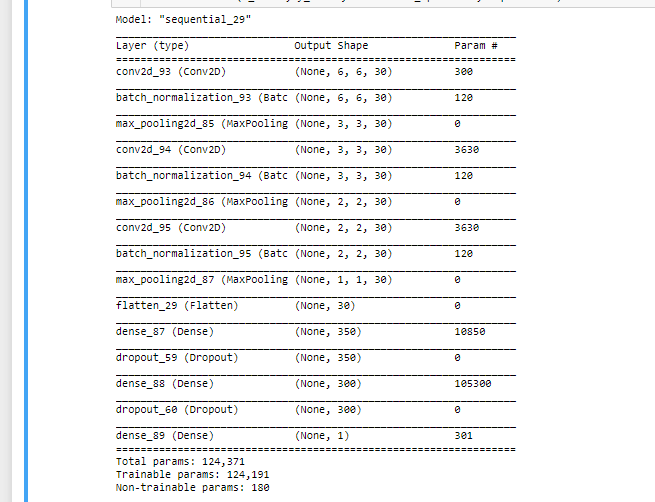


Figure 2: Base Model Summary

**Results**

We have experimented with the CNN base model by tweaking different parameters of the Conv2D, dense and max-pooling layers and by also increasing and decreasing number of layers to see how the outcomes are affected. In this section we are going to talk about the design and outcomes from our experiments with the network. We will mention only the circumstances where we have found the most significant changes in outcomes.

1. Base Model

We will start by briefly describing the architecture of our model. It is a sequential model consisting of three Conv2D layers with all of them having 30 filters. The kernel sizes are (3,3), (2,2), (2,2) for layer 1,2 and 3 respectively. There are also three dense layers with 350, 300 neurons in layer 1 and 2 respectively and 1 neuron at the output dense layer with the Sigmoid activation function since this is a binary classification problem. Each Dense layer is followed by a dropout layer to prevent overfitting. Each convolution layer is also followed by a max-pooling layer and a batch-normalization layer. We trained the base model for 60 epochs and the results we obtained are shown below:

Training Accuracy = 0.9752

Training loss = 0.06306

Validation Accuracy = 0.87867

Validation Loss = 0.5673

Test Accuracy = 0.8795

Test Loss = 0.5673

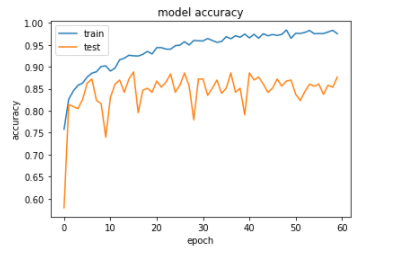


Figure 3: Base model accuracy

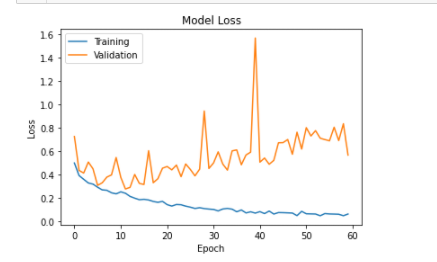
****

Figure 4: Base model loss graph

1. Increased Filters and reduced stride on Con2D layer

Keeping the dense layers identical to the base model, we changed the number of filters in the Conv2D layers to 100, 50 and 30, hoping to see some significant change in the output of the model. We changed the strides of the kernel of layer 1 to (2,2). The accuracy and loses are listed below:

**Training Accuracy = 0.9966 ^**

Training loss = 0.0145

Validation Accuracy = 0.9442

Validation Loss = 0.3950

**Test Accuracy = .9414 ^**

**Test Loss = 0.239 ^**

We observe significant rise in Training accuracy and drop in validation loss. The Accuracy on the test set also increased to .9414 which is the highest among all our experimentation.

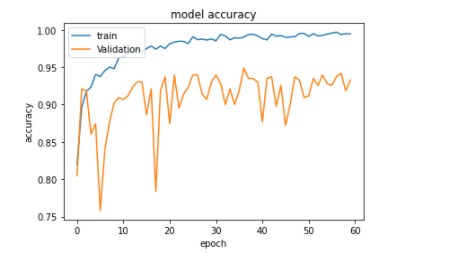
****

Figure 5: Increased filters and reduced strides in Convolution layer: Accuracy

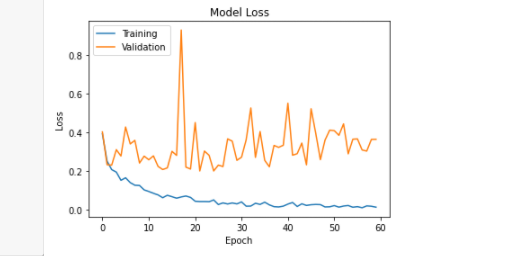
****

Figure 6:Increased filters and reduced strides in Convolution layer: Losses

1. Increased number of Convolutional layers

In this model, we changed the layer filters to 100, 50 and 30 for Conv2D layer 1, 2 and 3 respectively. We also add another layer with 10 filters and keeping everything else identical to the previous model. We obtain the following results:

**Training Accuracy = 0.9987 ^**

**Training loss = 0.0042 v**

**Validation Accuracy = 0.9233 v**

**Validation Loss = 0.5450^**

**Test Accuracy = .9330 v**

**Test Loss = 0.35 ^**

We can see that the model overfits on the training data too much and as a result we see that the validation accuracy and test accuracy drop significantly. Hence, we decide to not use this model as well.

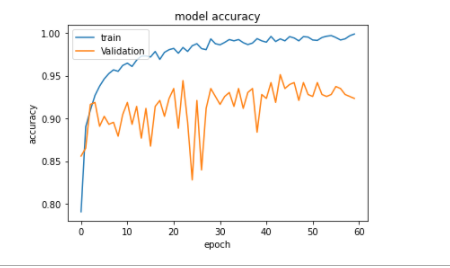
****

Figure 7: Increased number of Convolutional layers: Accuracy

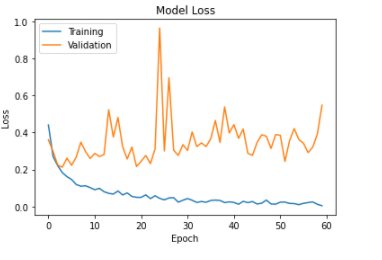
****

Figure 8 Increased number of conv2d layers: losses

Among all our models, the second model yields the best results. We use this model for classifying our ROI in the application.

**Conclusion**

While trying to increase our model performance on the validation and test set, we observe that:

1. Increasing dropout rate from 0.5 to higher does not reduce overfitting significantly.
2. Training the dataset for less epochs keeps training accuracy at satisfactory levels, but model performs poorly on test data.
3. Increasing number of epochs increase the validation loss and overfitting

The probable solution to the problems in the context of our project could be increasing the dataset i.e. making the model learn on a wider range of images to help with recognition. This can be a lengthy, time consuming and expensive undertaking. A way around this would be to perform Data Augmentation, which is to use the existing data to generate new image data for the dataset, usually done by adding noise to the images, rotating or manipulating the images in other ways and appending them to the training set.

**References:**

[1] Crash Stats: Drowsy Driving 2015, [online] Available: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812446>.

[2] parnec.nuaa.edu.cn/\_upload/tpl/02/db/731/template731/pages/xtan/ClosedEyeDatabases.html[3] G. B. Huang, M. Ramesh, T. Berg and E. Learned-Miller, **Labeled faces in the wild: A database for studying face recognition in unconstrained environments,** Tech. Rep. 07-49, University of Massachusetts, Amherst (October 2007).

[4] Wanzeng Kong,1 Lingxiao Zhou,1 Yizhi Wang,1 Jianhai Zhang,1 Jianhui Liu,1 and Shenyong Gao, “A System of Driving Fatigue Detection Based on Machine Vision and Its Application on Smart Device”.

[5] Fouzia, Roopalakshmi R, Jayantkumar A Rathod, Ashwitha S Shetty, Supriya k Alva’s Institute of Engineering And Technology, Shobhavana Campus, Mijar, Moodbidri ,” Driver Drowsiness Detection System Based on Visual Features”.

[6] Shinfeng D. Lin Department of Computer Science and Information Engineering National Dong Hwa University Hualien, Taiwan, R.O.C. david@mail.ndhu.edu.tw Jia-Jen Lin Department of Computer Science and Information Engineering National Dong Hwa University Hualien, Taiwan, R.O.C. [610121001@ems.ndhu.edu.tw](mailto:610121001@ems.ndhu.edu.tw)) ,“ Sleepy Eye's Recognition for Drowsiness Detection”.

[7] Xue Liu, Xiaoyang Tan, and Songcan Chen, “Eyes Closeness Detection Using Appearance Based Methods”.

[8] Belal ALSHAQAQI; Abdullah Salem BAQUHAIZEL; Mohamed El Amine OUIS; Meriem BOUMEHED; Abdelaziz OUAMRI; Mokhtar KECHE,” DRIVER DROWSINESS DETECTION SYSTEM”.

[9] 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, 2001.

[10] T. Danisman, I. M. Bilasco, C. Djeraba and N. Ihaddadene, "Drowsy driver detection system using eye blink patterns," Intern. Conf. on Machine and Web Intelligence, Algiers, Algeria, Oct. 2010, pp. 230- 233.

[12] L. Li, Y. Chen and Z. Li, "Yawning detection for monitoring driver fatigue based on two cameras," 12th Intern. IEEE Conf. on Intelligent Transportation Systems, St. Louis, USA, Oct. 2009, pp. 1-6.

**What we have learned**

**Ayman Mahmud Haque**

I have become familiar with the intricacies of the Convolutional neural network. I understand with more clarity how the layers interact with each other, how changing the different parameters change the outcomes at the different layers. I have come to know how the filters and kernel sizes in Conv2D layers affect in the extraction of features and patterns. I can visualize how max-pooling reduces dimensionality and how batch normalization scales the dataset. I have developed a comprehension of how to make the model best suited to different kinds of problems or manipulate a model to extract the features I need extracted. I have also become sufficiently accustomed to using open-cv and its various image processing functions.

**Vasavi Lankipalle**

In this project I have come across many new things. Researched through different approaches that have been solved for driver drowsiness detection system in the recent times. Firstly, searching for the right dataset was challenging. Learnt how to handle with large amount of real time datasets and to preprocess them. Learned how to make use of OpenCV functions like Haar Cascade Classifier to detect face and eyes from the image. Then experimenting with the Conventional Neural Network (CNN) model for getting better accuracy and results was interesting.

**Rahul Reddy Gadekal**.

This project helped me learn a lot of new aspects such as dealing with real time data, different parameters of CNN. We even have gone through different projects while preparing for the project, like how they use different classifiers and algorithms for the latest face detection technologies such as Ada Boost classifier, Local Binary Patterns (LBP), Histograms of Oriented Gradient (HOG), Support Vector Machine (SVM) and more. We even tried a lot a way to achieve our accuracy by tweaking our model with different parameters Conv2D, dense and max-pooling layers. We even experimented by adding and decreasing the layers to fluctuate the design and the outcomes of the project for better output. We had to spend a lot of time on data acquisition and processing since we didn’t have the exact dataset we needed, so we had to preprocess the data that we got as per our requirements.