Drowsiness detection using CNN and Android application

Submitted to the Teesside University as the dissertation project for the degree of MSc Date science

School of Computing, Engineering and Digital Technologies

Department of Computing and Cybersecurity

Teesside University

Middlesbrough

TS1 3BA

Date : 11th January 2022

SUBMITTED BY

Arun Sai Pilli (w9323581)

W9323581@live.tees.ac.uk

School of Computing, Engineering & Digital Technologies

TEESSIDE UNIVERSITY

SUPERVISOR

Dr Zia Ush Shamszaman

Z.Shamszaman@tees.ac.uk

Senior Lecturer in Computer Science

Department of Computing and Games, TEESSIDE UNIVERSITY

Abstract

In the automotive industry, many breakthroughs are made to make road driving as safe as possible; most of these improvements, such as autonomous vehicles, use computer vision technology to prevent accidents; however, few advancements are made to account the physical circumstances of the driver. Driver drowsiness stands out when all types of traffic incidents are studied. This is a scenario that occurs all over the world. Thanks to developments in computer vision and the use of CNN, we can identify drowsiness in drivers. We use deep learning to construct a model that detects driver tiredness based on eye ratio in this study, and then implement the same concept in an Android app. We'll use Google's Machine Learning Kit APIs to extract the face and eye regions, as well as estimate the eye ratio for drowsiness detection from then ROI and alert the driver with an alarm, our model performs better and will be able to show some good results on the both platforms.

Table of Contents

Table of Figures

Acronyms

Acronyms	Description
CNN	Convolutional neural network
ML	Machine learning
OpenCV	Open-source computer vision Library
FD	Face detection
ROI	Region of Interest
IP	Image processing
EAR	Eye Aspect Ratio
CV	Computer Vision
CPU	Central processing unit
GPU	Graphic processing unit
GD	Gradient vector
DP	Displacement vector
RTI	Real time implementation

1. Introduction

According to the National Highway Traffic Safety Administration (NHTSA), sleepy driving is responsible for around 2.5 percent of all fatalities in car accidents (Chirra, ReddyUyyala and KishoreKolli, 2019). in which driver drowsiness is responsible for around 1.4 million deaths each year, making it the seventh leading cause of road accidents worldwide (Pattarapongsin et al. 2020). This type of accident is dangerous not only for the driver, but also for passengers in the same car and nearby automobiles. Many factors contribute to a sleepy driver, including a long travel and driving without resting for an extended period of time. This scenario is prevalent among heavy vehicle drivers, who, in their quest for more money and time savings, drive for longer hours, putting everyone at risk. Road developments and transportation systems have improved dramatically in every country as a result of engineering and technology advancements, and many advancements have also been achieved in automotive safety features that help passengers save lives when accidents occur.

Artificial intelligence's growth and implementation have skyrocketed in recent years, owing to the fact that it is open source and contains simple methodologies that may be used to solve any real-world problem. Leading automotive firms such as Tesla, BMW, Mercedes-Benz (Jabbar et al. 2020), and others have used AI to develop some astonishing breakthroughs in automobile safety, including self-driving cars, lane change alert, auto parking, and many other features that have helped minimise accident risk. Although these advancements are not totally self-contained, they do require drivers' assistance to rectify their operations from time to time, and these methods are undertaken to ensure the safety of drivers in the event of an accident and mostly given to the high-end vehicles. The primary goal of safety is to prevent accidents, and one of the most important features is to watch and analyse drivers' physical activity in order to monitor and apply AI.

Physiological and behavioural measures are the most common ways for identifying driver drowsiness. Physiological measures like wearable, lightweight brain sensing headbands, electroencephalogram (EEG) (Ceamanunkul and Chawla 2020) and electrocardiography (ECG) can be used to assess drivers' physical status and provide accurate data, but they are too bulky and expensive to be used. Behavioural measures, on the other hand, are the best technique to identifying drowsiness using deep learning models, and these models can be deployed in an Android application, which can then be easily loaded in any automobile. The dashboard camera and cutting-edge deep learning techniques were used.

In this research, I propose a drowsiness detection model that is implemented in four steps. To begin, facial areas and landmarks are retrieved from the video frames captured by the webcam as input. Second, a sequential model with labels is trained over a dataset (opened and closed eyes) Once the models have been fully trained and evaluated, they are saved with newly updated weights and bias, which can then be used in the future. Third, by using a webcam on a computer, it can be utilised to classify sleepy and non-drowsy states. We'll move on to the Android application once the model's performance has been evaluated and validated. Fourth, we'll use Android Studio to develop a basic app with Google ML Kit libraries and our model structure for main function in java files, which we'll then validate.

2. Research question

What are some deep learning methodologies for using a CNN and an Android application to detect drowsiness in a person?

2.1 Research focus

For this project, I looked into the machine learning and deep learning techniques that are used in face detection and ROI, as well as the CNN models that are used for classification, and then I looked into OpenCV and google open-source and why it has been so popular in recent years. After reading several articles, literature papers, and journals, I found a few techniques that are currently being used and successfully implemented in the real world. I then researched those technologies using various sources and came up with a simple idea to implement this project on the CPU and in an Android application.

3. Literature review

Drowsiness detection was seen as an essential element in road accidents, and it is an area in which study might be undertaken to identify potential improvements and lessen this factor causing these types of accidents. Many automobile companies and researchers have developed various techniques using various methods to create high accuracy solutions based on a variety of factors and conditions that cause drowsiness and its effects in and around automobiles, such as weather and road conditions, vehicle GPS monitoring, drivers brain waves and habits, and heart rate while driving, among others. Traditional methods are regarded to be the most cost-effective because they collect data with heavy machinery, as discussed in the sections below. Most techniques are based on algorithms that use statistics and machine learning concepts. Because AI is open source, many algorithms and techniques are created on a daily basis with the addition of new factors alongside traditional methods. When we consider all techniques, we can classify them into three different categories, which are discussed in detail in the section below.

The first approach is based on vehicle behaviour; in advance, each vehicle feature is equipped with sensors that detect and record data; analysis of the data can provide a wealth of inputs for drowsiness detection. Using data from vehicle features (McDonald et al., 2018) published an article in which he proposed a temporary and contextual algorithm that uses vehicle driving components such as steering wheel, acceleration pedal points, and vehicle speed limit. These measures are fed into a Dynamic Bayesian Network to determine if the driver is drowsy or not. In comparison to PERCLOS (United States, 1998) which make predictions based on eyelids and its moment, the model was able to yield some noteworthy results with substantially lower false-positive outcomes. These parameters results may vary in highways where constant driving and non-use of features can produce low false positive results and make correct detection difficult; however, this algorithm output can be combined with others to develop new algorithms for accurate drowsiness detection in any location. (Yu et al., 2019) mentioned a study in his work that was based on an investigation of tiredness when changing lanes, and it demonstrated the effect of drowsiness after being given alert when changing lanes (Rimini-Doering et al., 2005) has examined a study of driving behaviour as driving speed changes Even he was unable to obtain good detection findings; all these methods are difficult to test practically and yield low accuracy.

A second method is through Physiological Based Methods; in this category, the driver's drowsiness is detected using Physiological Based Methods such as Electrocardiogram (ECG (Zhang, Wang and Fu, 2014), Electrooculogram (EOG) (Xue-Qin Huo, Zheng and Lu, 2016), Electromyogram (EMG) (Khushaba et al., 2013), and Electroencephalogram (EEG) (Wang, Li and Liu, 2018). These physical gadgets aid in understanding human body behaviour while drowsy; data generated by these approaches is highly accurate because they are directly connected to the human body and sent to algorithms to evaluate sleepiness output. However, these old methods are highly expensive and require a large amount of room to set up these equipment's. Because they are large in size and cost a lot of money, most of them are limited to lab research and cannot be used in a practical life.

Three techniques are based on computer vision; huge breakthroughs in deep learning technology have provided tools for classification and detection in computer vision; its applications are employed in a variety of industries. Agriculture, medical, health care, object detections (Kamilaris and Prenafeta-Boldú, 2018), very high application of this technology in image data recently witnessed due to increase in media data (Ortiz-Ospina 2019) how much data/ I have searched] Keeping accurate data has been the most difficult task; to manage this data, the number of apps that use image data has expanded dramatically; and because computer vision is an open source platform, many contributions have been made in image processing, Our model falls into this category since we employ an image processing technique to detect driver tiredness. Drivers' facial expressions and eye lid movement vary when they are drowsy, thus many scientists have taken advantage of these changes and attempted to provide solutions and models using deep learning approaches to identify drowsiness in drivers.

(Said et al., 2018) created a drowsiness detection model based on eye detection. In this work, viola jones models were used to detect the driver's tiredness based on the facial and eye region and alert the driver via an alarm. The accuracy was only 72.8 percent outside and 82 percent indoors, respectively (Sooksatra et al., 2018) In this study, viola jones algorithm is used to extract face region and to extract expressions from it he used Gabor wavelet decomposition and for feature selection he used Adaboost learning algorithm and finally he used HMM for classification of drowsiness expression. Based on the above two models, (Chirra, ReddyUyyala and KishoreKolli, 2019) implemented a drowsiness detection project using Viola-jones algorithm (face detection and feature extraction) and passed on to CNN model with four convolutional layers and soft max layer to classify drowsiness images, 96.42 percent accuracy was achieved with this model, but did not specify any drawbacks, and suggested to use transfer learning in future improvements.

(Sooksatra et al., 2018) proposed a drowsiness detection model based on visual features in a video series, with lighting circumstances no longer influencing model performance. He offers an eye detection method based on normalised cross-correlation between displacement and gradient vectors. Gradient vectors are made independent of illuminations by normalisation stages and detect human eye. Infrared auxiliary illumination is also employed to make the model work in all conditions. (R, Goraya and Singh, 2018) proposed a model based on EAR by image processing application with live video, Dlib functions are trained using HAAR cascade algorithms to plot facial landmarks in a trained neural network and detected drowsiness using the eye aspect ratio. if the blink is longer than standard rate the drowsiness is alerted by an alarm. (Pattarapongsin et al., 2020) This technique employs a combination of DNN, CNN, EAR, MAR, and face posture. First, the DNN model detects the face and crops the image by detecting only the face; next, CNN extracts 98 particular spots on the face, and EAR, MAR, and face pose are calculated to arrive at a drowsiness conclusion. (Tayab Khan et al., 2019) proposed a model based on angle of eyelid curvature to detect drowsiness and achieved 95% of accuracy but failed to perform on lowlight conditions.

Mobile Net-SSD architecture is used by (Shakeel et al., 2019) to train on few images to achieve mean average precision of 0.84, This model was so efficient that it was deployed in an Android application and in any camera streaming to make real-time classification judgments. A combination of eye, mouth and pose of head lean are consider simultaneously in (Khyat et al., n.d.) to detect level of drowsiness, this study was conducted on NTHU (Weng, Lai and Lai, 2017) dataset. A study was conducted by (Xie, Chen and Murphey, 2018) employed the NTHU-DDD and YAWDD databases on a transfer learning and sequential learning solely on a yawn clip to get a higher precision rate and accuracy to the change in head angle position. (Jabbar et al. 2020) proposed an improved CNN based machine learning model to overcome their previous machine learning model that classifies images using a multi-layer perceptron-based model [D2MLP-FLD] due to model size and performance. Their previous model, which was 100kb in size and could produce 81 percent accuracy, was upgraded, this paper in particularly helps us to understand how to improvise the model size and performance.

(Ceamanunkul and Chawla, 2020) A machine learning model was proposed to identify tiredness using facial points and face emotions (pre-trained convolutional neural networks are utilised for emotion identification), and the two vectors were combined and supplied to PCA as input by decorrelated. (Biju and Edison 2020) They proposed a drowsiness detection system based on two real-time datasets using deep convolutional neural networks, using YOLO for facial detection and the Inception-v3 pre-trained model for classification of driver drowsiness. They claimed that this model could be performed without the use of powerful GPUs. To detect drowsiness (Yu et al., 2019) proposes the DrowsyDet framework, which may be integrated into mobile applications. To obtain the drowsiness result, three CNN models are built to extract and classify face, eyes, and mouth regions, and these values are given to the DrowsyDet model, which then computes these values parallelly and output is evaluated with an equation mentioned in the paper to obtain the drowsiness result. Furtherly they proposed mobile application model which can be created using java and android API, which contain the same units as the trained model. (B et al., 2021) worked on a proposed model using two architectures, VGG16 and CNN; basic facial landmarks are obtained by CNN and EAR, MAR features are obtained by VGG16; both models achieved more than 95 percent accuracy with the state of the driver mentioned constantly; however, a new model was introduced in the work. (Nguyen, Putro and Jo, 2020) The optimal number of parameters and computations in the network make this model lightweight and portable to deploy in and platform, as other papers included in the previous study using facial detection and eye status are classified with the network and classified the output as drowsy or non-drowsy, as other papers included in the previous study using facial detection and eye status are classified with the network and classified the output as drowsy or non-drowsy. (Mansur and Shambavi, 2021) The object detection training is done using an open-source model, and the framework is constructed as a standby unit in a vehicle by employing facial region and ROI (Region of Interest) to anticipate the outcome. Keras and TensorFlow are used as backends, while the Raspberry Pi model 4 camera module V 1.3 is used as a backup. standalone hardware is uploaded with AI training, and able to predict with 95% accuracy.

By reviewing the previous literature, we have discovered that deep learning-based models are mostly implemented and successfully executed in the real world. To detect drowsiness, there are a few factors that need to be taken into account before we build a model, such as the driver wearing a mask, sunglasses, head tilt, yawning, and light conditions. By keeping all these features in mind, we propose a model that will detect drowsiness only based on eyes ratio using Haarcascade feature extraction and deep learning architecture to classify whether driver is feeling drowsy nor drowsy, and the same procedure is followed in the developing android application where we use google ML kit API'S to perform face recognition, ROI, and drowsiness detection.

4. Prerequisites

Before we move on to the main methodology, there are a few prerequisites that must be met in order to comprehend this project. From a knowledge standpoint, I'll give a brief overview of each deep learning concept below that are required in this project, and for the coding prerequisites we need intermediate python skills.

4.1 Hardware requirements

For this project, a standard CPU will suffice; a GPU is not required, nor is it mandatory, as it is for other deep learning models; however, you can use it for faster computation; however, you will need to set up a separate setup with your GPU to work with TensorFlow: below is the screenshot for preferred system configuration for this project,

Figure 1. Preferred CPU configuration

4.2 OpenCV

Figure 2. OpenCV logo

OpenCV is a free open-source programming library for real-time computer vision (Minichino and Howse, 2015). Computer vision is a branch of computer science that focuses on images and videos. Because we live in an age of video and images, computer science is the study of these technologies using computers. Intel was the first to develop OpenCV as a project. It's a cross-platform library that

supports python, java, C++, and other programming languages, and because it's an open-source platform, it works with a wide range of platforms, CPU architectures, and devices. It's one of the most extensively used library packages for video detection, motion detection, video recognition, image identification, and deep learning facial recognition frameworks.

To assist some of these fields, OpenCV provides a number of strong application cases, including 2d and 3d feature toolkits, facial recognition systems, gesture identification, human-computer interaction, motion understanding, and object detection, to mention a few. Machine learning libraries are included in OpenCV, which contain numerous algorithms such as k nearest neighbours, random forest, and nave base, among others. These are the major strengths of machine learning algorithms, and they are necessary in computer vision applications.

OpenCV's basic objective is to produce open-source optimised code that can operate in real time and re-use its modules rather than re-creating them. The code is easily legible and transferable, allowing it to work in any platform and enabling others to create and share it in open-source platforms. OpenCV is built in C++, and its principal interface is also written in C++. It also has a number of interfaces that can be used to support other languages. That is, the C++ interface may be implemented in Python, allowing us to use OpenCV in the Python language. As a result, we're using it in our model, and we're installing it with **Pip install OpenCV**

4.3 TensorFlow

TensorFlow is a mathematical computation library for training and building our machine learning and deep learning models with simple to use high level API's (Pramod Singh and Avinash Manure, 2020). As an example, If we create a machine learning model in SPARK, SCIKIT-LEARN, or R library and use it to infer for millions and billions of data points, the output of logistic regression is shown below, which is nothing more than a mathematical formula in which your weights are multiplied by all the variables and corresponding coefficients, and then you add your intercepts to get an output, which is then passed to the sigmoid function to basically covert that into a probability.

Figure 3. TensorFlow logo

From the diagram below, we created a computational graph with TensorFlow, in which I have an X variable, which is my input variable matrix, and a weight matrix, which is nothing more than a

coefficient for each variable. When performing matrix multiplication, add intercepts and use a sigmoid function. Now that this computational graph has been built, rather than inferencing on a single data point, we can schedule this computation graph to run on GPU or CPU. This is simply one aspect of it, but how does it vary because we are using NumPy? We can now take this data and put it in a GPU memory and vectorize it so that we can use all the GPU cores or TPU cores.

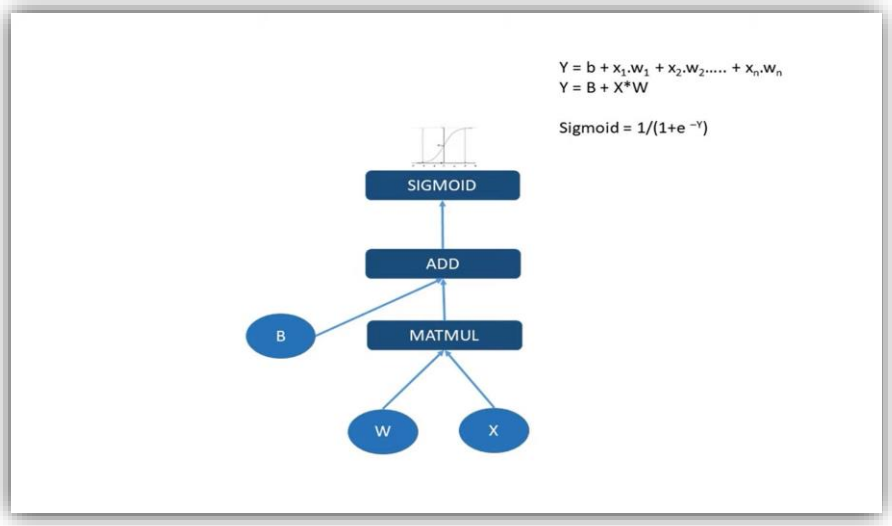


Figure 4. Computational graph in TensorFlow

Essentially, we are breaking down the formula and executing it on multiple hardware accelerators to speed up my computation. Consider a complex formula with many equations. You can take the breakable unit and schedule it on multiple GPU or CPU accelerators. Essentially, we are generating a computational graph, dividing it down into chunks, and then executing each chunk in a specialised hardware accelerator, vectorizing it to use the core of the hardware accelerator.

TensorFlow is straightforward to deploy across numerous systems because it runs on mobile devices, PCs, servers, and even web browsers. TensorFlow also supports numerous distribution strategies, allowing us to distribute machine learning models, deep learning models, and mathematical functions across a variety of devices. TensorFlow installation requires a specific hardware requirement, please review the system requirements on the TensorFlow website to ensure that your computer fulfils the requirements for TensorFlow installation.

4.4 Keras

Keras is a Python-based deep learning API (Patidar, 2019) that runs on top of TensorFlow, a machine learning platform. It was created to allow for quick end-user experimentation, allowing us to move quickly from concept to implementation. There are a few other neural network APIs that compare to keras, but keras makes learning how to use libraries and their features fairly simple, making it perfect for beginners. Knowing Keras also aids you in dealing with other small syntactical computational problems that may arise throughout the implementation of this project.

Figure 5. Keras Logo

Keras was recently integrated with TensorFlow, but before that it was a high-level neural network API that could be configured against one of their separate lower-level API's such as TensorFlow, Theano, and CNTK, and later it was fully merged with TensorFlow API and no longer a separate library, and we are only using these high-level keras API's without making much use of the lower-level TensorFlow API's. Because it is fully integrated with TensorFlow, keras can be quickly installed into our machine. The installation method is simple: simply type **pip install TensorFlow** from your command prompt project environment.

4.5 NumPy

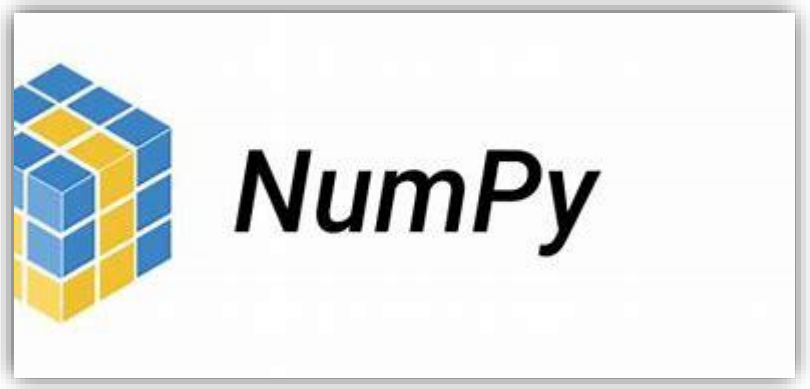


Figure 6. Numpy logo

NumPy (www.youtube.com, n.d.) is the abbreviated version of numerical python; it is a commonly used library in Python, particularly when dealing with numbers; it is significantly faster than any other library in computing numbers. It has a number of built-in mathematical functions that allow us to manipulate data, from constructing multi-dimensional arrays to accessing its elements and performing mathematical operations on them. NumPy is used to generate multi-dimensional arrays to calculate the matrix. Matrix is a mathematical notion that we use in statistics and graphs, as well as image analysis such as converting the motions of 2d and 3d images, all these are a used cases of having multi-dimensional array towards matrix. Installation process is just like other package installation as shown

in the above topics, which is **pip install NumPy** in the project environment which we created for this project.

4.6 Android Studio

For android application development, I have used Android studio because it is simple to install and use for app development. There are many other android development environments and websites out there that will provide the same functionality as this one (freeCodeCamp.org, 2020), but I found android studio to be simple to use, and there are many videos on YouTube that explain the functionality of this application and application development tutorials. Before we install the Android Studio, we must first ensure that we have Java installed on our computer. This can be done by typing java -version in your command prompt to check the version and availability of Java. If you don't have Java installed, there are many videos on YouTube that will show you how to do so. Secondly, we need go onto the android studio website link given below and download and follow the installation process, I have used the 2020.3.1 studio version on Windows 64-bit, which is 0.9 GB in size. This version is freely available on the internet, and because Android is an open platform, it is free.

Download Android Studio and SDK tools | Android Developers

4.7 Android phone

For this project, I used a Samsung android phone, but any android phone with SDK version 6.0 and above will work perfectly in Android Studio. To connect the android phone, we can use either a cable connection or an Android virtual device to develop and test our code. If we are connecting via cable, we must enable our android developer option setting through phone settings.

We've created a new environment in Anaconda for this project that only has the necessary files and libraries, or dependencies installed. Using the PIP installation, we'll be able to install all the above libraries in the project environment. If any dependencies are required to support these libraries on your machine, they can be found in my requirement.txt file, which contains all the project's dependencies.

5. Dataset and pre-processing

We primarily need image data for this project that contains images of humans with clear eyes open and closed for model training purposes; we can either choose a dataset that is readily available in Kaggle, or we can create our own dataset; however, in keeping with the ethical conduct code, we have decided to go with the dataset that is readily available in Kaggle; the link is provided in the below description.

yawn_eye_dataset_new | Kaggle

Figure 7. Dataset according to labels

This dataset contains 2900 images of 100 distinct persons, each of which is labelled with one of four categories: yawn, no yawn, closed eyes, and open eyes. We considered all the three labelled present in the dataset for training the model and using it in future works or advances because we just need closed and open eye data. This dataset was downloaded to my local disc and utilised in my project by calling the read method with the path. This collection contains only coloured images at a dimension of 480 x 640 pixels. To clean up data I manually combed through each image to remove any photos that were not relevant to the labels stated, we achieved satisfactory results working on this project, therefore I only provided the image after pre-processing. If the model performed poorly and the results were not up to expectations, I would have included another dataset or more images to train the model, but I used a data augmentation function called image generator in the model, which provided more image data to train the model, which I have explained in detail in the section below.

6. Proposed method

This project is carried out in two different sections. The first section tries to implement the proposed drowsiness detection model in a laptop using Haar-cascade for feature extraction and face detection, and then a CNN model is built using dataset imported from Kaggle to train and classify the input as drowsiness or non-drowsiness by alerting the driver with an alarm, and the second section builds an android application in android studio using java and google ML Kit APIs for drowsiness detection.

7. Drowsiness detection using CNN model

We will use OpenCV to provide data from a webcam and feed it to a deep learning model (CNN) that will identify the driver's drowsy condition as open or closed eyes. We will be utilising Python to construct algorithms for drowsiness detection, below is the flow chart of the model.

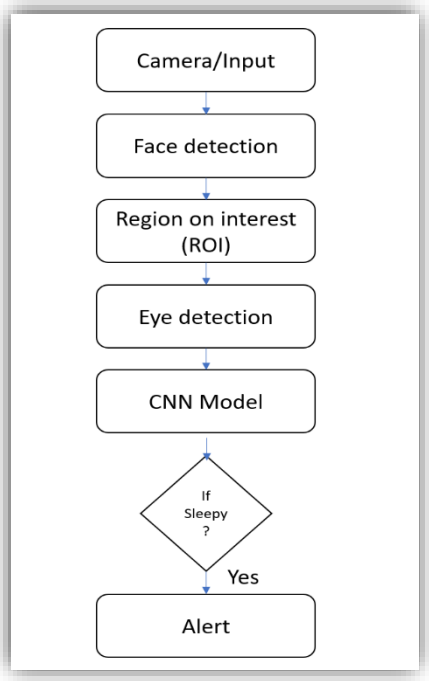


Figure 8. Drowsiness detection flow chart

7.1 Face detection

We are mainly focusing on eye regions for drowsiness detection but not the complete face recognition, therefore a reset of regions in the picture like backdrop and other features on the face is not required for our project. To achieve this, we need to learn a few concepts, one of which we are employing in our project is feature extraction. There is a plethora of algorithms that do these duties for us. However, the feature extraction revolution gained steam after (Viola and Jones 2001) published a paper in 2001 presenting a machine learning approach for feature detection that processed photographs quickly and accurately. This method is based on the Haar cascade files, which we used in this project. This algorithm is made up of four major components: a haar feature selection

algorithm,an integral image, an Ada boosting algorithm, and a cascade classifier. The algorithm's general flow is as follows

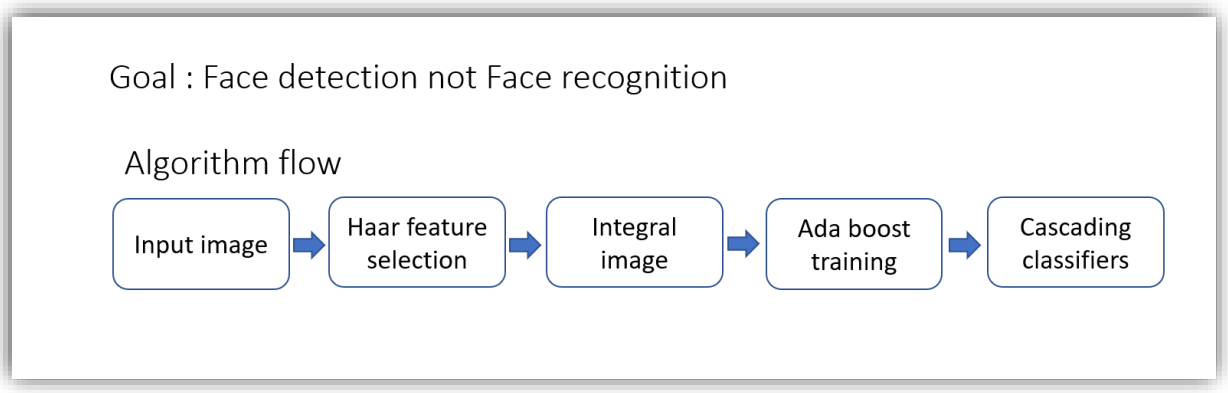


Figure 9. Haar cascade algorithm flow

7.1(a) Haar feature selection

At first, all of the images that are given as input are converted to grayscale colour, and then haar features are the rectangular features that are masked over an image within each rectangle summation is calculated, and then difference between black and white regions are calculated, for example, the background colour in the image is mostly of the same colour when these features are masked over them they give distinct results that do not indicate the facial features by calculation. When they are put over the face, they overlap the image and indicate haar characteristics, as illustrated in the image below. There could be (Viola and Jones 2001)180,000+ features in a 24x24 resolution image, but they are all useless for finding the features, and computing all of them is time consuming, thus viola and jones devised a solution, which is discussed in the next part.

7.1(b) Integral image

A integral image is created by adding the intensities of rows and columns before each and every pixel as shown in the figure below, from the original image when you calculate the feature from it by sum of the black and white side are calculated and they are subtracted from each other (Viola and Jones 2001), an integral image is created by adding the intensities of rows and columns before each and every pixel as shown in the figure below,

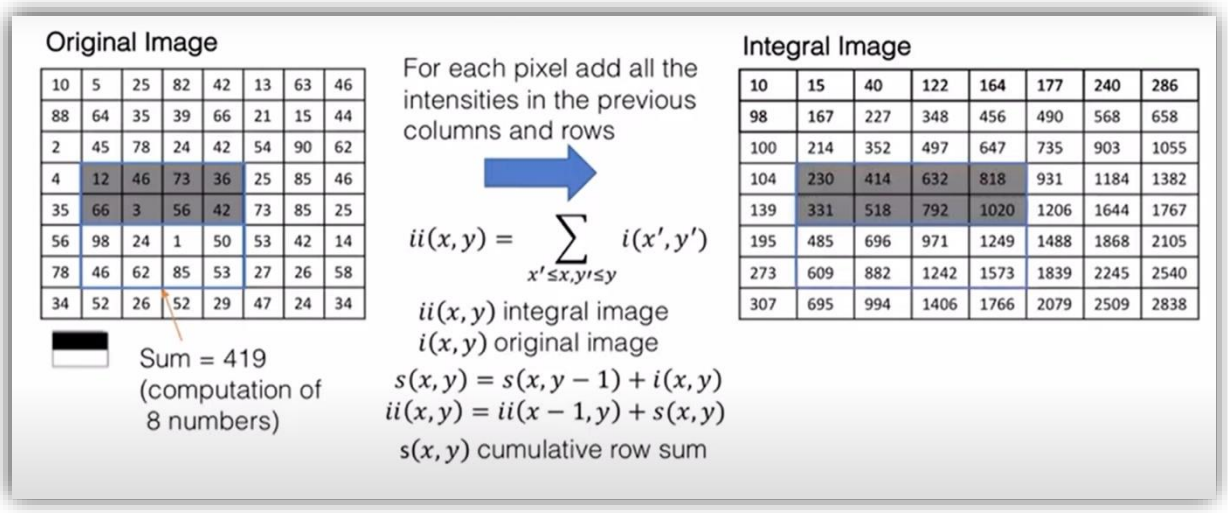


Figure 10. Integral equation

We can see from the above figure that all the pixel values in the integral image are created by adding each pixel to the previous pixel, and the formula is given in between to show how it is done. To calculate the sum, when we have haar features in the pictures, such as the one highlighted in dark colour in the below picture, we only need to operate next to the corner of the pixels as shown in the below picture.

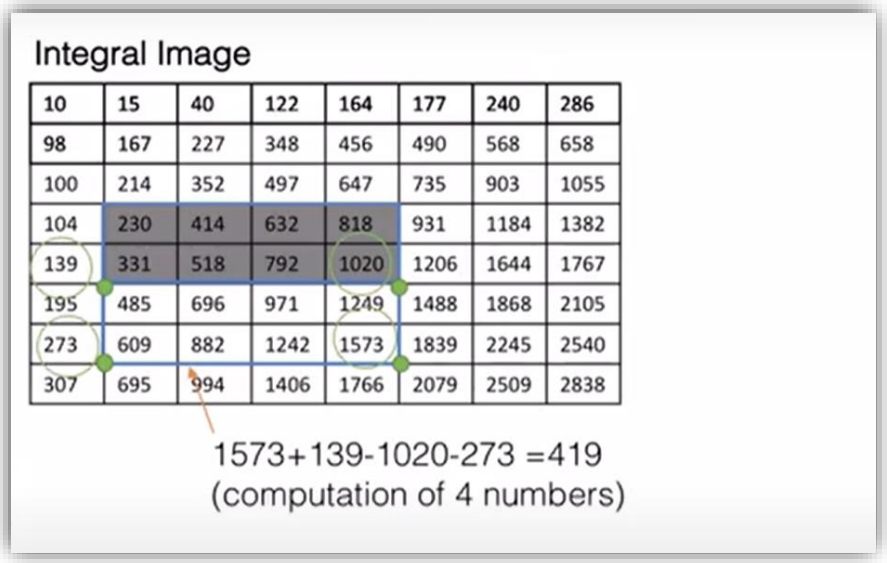


Figure 11. Integral Image calculation

From these calculations, operating on 100x100 photos would require only 4 operations instead of 10,000. With this integrated image, the Haar feature computation is considerably reduced, and the image may now be scaled to scan the image rather than resizing it for feature detection.

7.3(c) Aba boosting

As previously said, not all Haar features are employed; therefore, in order to speed up the process and obtain correct results, we must train the features on photos to use the proper features in the right places. This may be accomplished by supplying a large number of images to the algorithms. To distinguish between the two types, viola and Jones employed 4960 facial photos and 9544 non-facial images in their algorithms (Viola and Jones 2001). They manually entered this is the face and this is not the face into the system.

Because not all the useful qualities are equally important, they proposed a strong feature that combines weak features with their appropriate weights. Because it can detect a face in addition to other things, the feature is limited. However, it is the combination of the features that makes them powerful. So, we may identify the weights of the system facial images, which indicate that they are positive examples with a value of 1, and non-facial images, which indicate that they are negative examples with a value of 0. The weight of the classifier in the images is now normalised, and a classifier with one feature is employed and trained on all images, and the error is computed; if the face is detected on a facial image, the error is zero; otherwise, the error is 1, as is the case for non-facial images. The inaccuracy is then compounded by the image's relevance. As a result, the weights are changed to reflect the smaller inaccuracy. As a result, the image's weight is adjusted for the following iteration by reducing it for the image that was successfully categorised, and the error is assigned a higher value in the strong feature.

Finally, the final strong correctly classifier is one when the sum of the weighted features is higher than the half of the sum of the weights.

7.4(d) Cascade classifiers

Following ada boosting training, it was discovered that a characteristic on the eye and chins, where the eyes are darker than the chins, is the most important, while the second signals that the bridge of the nose is brighter than its surroundings. Cascade the classifiers to see if a window includes a face feature quickly. The first classifier is the one with the highest weight, which was discovered earlier. Non-facial features will be rejected outright; if the first feature is approved, the second classifier will be used until all the features have been approved. Then a face is discovered. This is far more efficient than trying all the face detection options.

In brief, the Viola Jones algorithm will take a grayscale image, integrate it, open a window, and slide it across the image, and then do cascade classification with the previously obtained feature for each window (Viola and Jones 2001). A face will be detected, and a new window will be examined if all the features are accepted. If this is not possible, a new window will be considered. A 384x288 pixel shot took the viola jones algorithm about 0.067 seconds to analyse. It was created in 2001 using a 700MHz Pentium processor. It was the world's first successful real-time facial detection system.

7.2 Region of interest (ROI)

As per the above theory, we have detected the face in a input image, at first we have converted our image into grayscale as OpenCV algorithm will only be taking grayscale images for face detections, and then we call the haar cascade files to detect face, below line is used in our model to set the classifier and detection.

face = cv2.CascadeClassifier('file location')

faces = face.detectMultiScale (gray)

It returns the array of detection with x, y co-ordinates, hight and width of the boundary box for the object, then the iteration will take place over the face by drawing a boundary box over each face.

f**or (x,y,w,h) in faces:**

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

7.3 Eye detection

For this section, follow the same steps as before. At first, we implement the cascade classifier for the both eyes and they are represented as Leye as left eye and Reye as right eye and then detection of eyes are done through following equation in the model, we are only extracting both eyes from the image, and this can be achieved by drawing the boundary box and extract the image from the frame using the below code and later these images are given as an input to the CNN model to predict if the driver is drowsy or not.

left_eye = leye.detectMultiScale(gray)

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

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

8. Drowsiness detection algorithm

a) Importing all the necessary libraries
b) Alert initiation
c) Loading cascade files to detect the face and eye marks
d) Labels are assigned to display on the cam window
e) Loading the CNN model (drowsiness_model.h5)
f) Starting the Webcam, detecting the face and placing rectangular frame
   cv2.rectangle(frame, (0, height-50) , (200,height) , (0,0,0) , thickness=cv2.FILLED)
g) Marking the Face, left and right eye using the Haar Cascade files,
   Converting the image into grayscale and resizing the image to (24x24),
   predicting right and left eye by calling the model
   rpred= np.argmax(model.predict(r_eye), axis=-1)
   lpred= np.argmax(model.predict(l_eye), axis=-1)
h) Defining the count of lables assigned, based on that alarm will be initiated less than 5secs to alert the driver. The accuracy is based on the Model predictor
i) If score> 7 then person is feeling sleepy so we beep the alarm

9. CNN model

CNN is type of neural network that has become highly popular in recent years for image classification and analysis, as well as other data analysis and classification processes. A typical ANN cannot handle this type of classification problems as it takes full-scale image as an input and end up with very high number of parameters which might end up in overfitting. This is where CNN model comes in picture, CNN is a type of specialised neural network that can identify and pick patterns and provide us with information. There are mainly three types of layers in CNN(Guo et al. 2020), **a convolutional layer, a pooling Layer, and a fully-connected layer.**

In a Convolutional layer, Pattern detection is the most important technique in image analysis, and it's all done in the hidden layers of the CNN's neural network. As illustrated in the diagram below, this convolutional layer has a collection of **filters (Kernal's)** that recognise patterns in a given image, pattern features in images such as edges, circles, texture, and object, and so on.

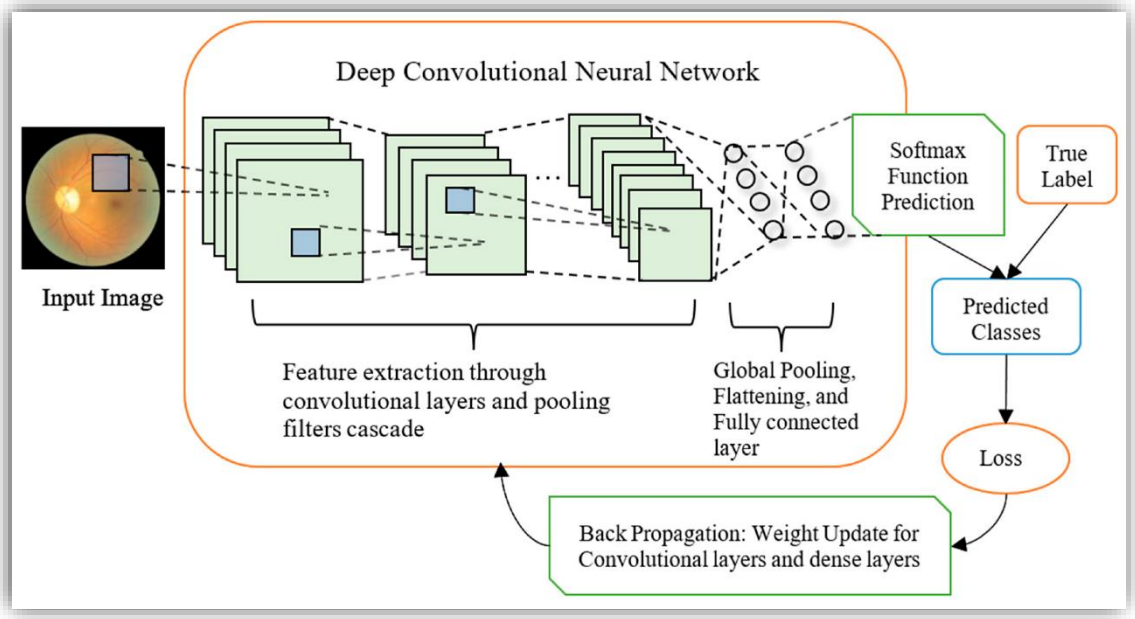


Figure 12. Feature layer and max pooling layer of CNN

Each pattern filter has its own functionality, and there are many in CNN. These are the network's initial layers, and the deeper our network becomes, the more sophisticated these filters become at detecting the most significant elements of the images, such as the eye, nose, and hair. For an example above in the example of a image with number seven on it and these of 3x3 matrix slide over our input image and receives its values in number which in term detects the patters which is also called features maps, are might be more than one feature maps in a CNN and each feature maps represents each feature in

the image, very important step in CNN is models will automatically will learn to use how many filters are needed for a given images, this is achieved by backpropagation in the network. once these features are extracted, they are flattened into one dimensional array and sent to **fully connected layer** for the classification of the image as yes or no.

In this section, we will build a CNN model to classify whether the given input from the previous step has a drowsy eye or not. There are a few topics that require some understanding to understand the working or functionality of the CNN model, and I will provide some brief descriptions of the topics as we go along while building and technical implementation of CNN model. We will build this model in a separate file, and with the help of TensorFlow inbuilt functions, we will be able to save the fully trained model and use it in drowsiness detection code.

9.1 Activation functions

An activation function in a neural network is the output of the neuron(Goyal et al. 2020) when it is given a set of inputs, When a sum of weights of inputs layer are provided to a single neuron in the hidden layer and passed that weighted sum to the activation function, the activation function makes some type of action to transform that sum to between some lower limit and upper limit. This activity is the exact function of our neural network in the human body; when a neuron receives a response, it makes a yes or no decision, which is represented same in hidden layer as 0 and 1, where 0 represents less activation and 1 represents great activation. There are different types of activation functions, the most popular in them are Relu (rectifying leaner unit) and sigmoid function.

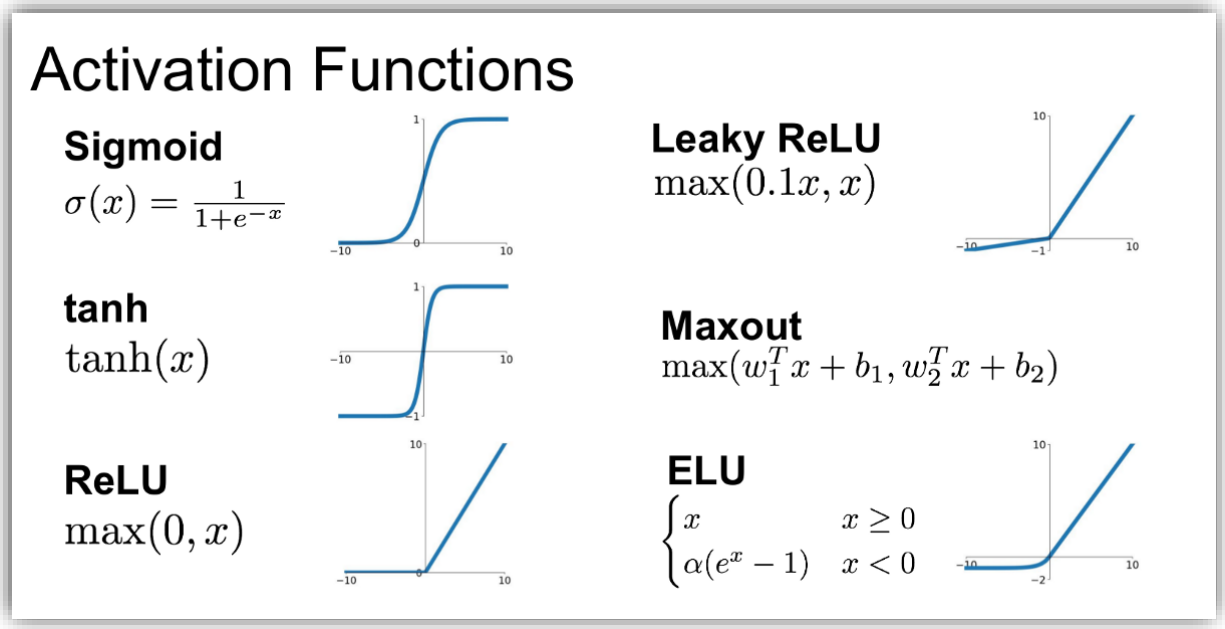


Figure 13. Types of activation functions

Relu: This activation function takes the input and transforms the value to either 0 or the input itself. Any value less than or equal to 0 then the Relu transform the value to 0 and any value greater than 0

stays the same. The idea of the activation function is more positive the neuron higher the activation it is. This type of functions is mostly seen in classification problems.

Sigmoid: When you take sigmoid function as a activation function then output of the activation function ranging between 0 to 1, sigmoid takes the input and transforms any negative number as close to zero and any higher number in positive transforms to close to 1, this type of activation function is largely seen in a multi- class classification the higher the output value is the accurate the outcome is.

9.2 Max pooling

It's a form of procedure that's implemented after each convolutional layer in CNN models to lower the image's dimensionality by reducing the pixels values in the preceding layer's by keeping higher values pixels in the output which in turn will helps to find most activated features by these large pixel values. We know that each convolutional layer has filters with specified dimensions (3x3) that convolve the input image into a new matrix of computational pixel image from the previous section. The image is then sent into our max pooling operation, which is a nxn corresponding filter that strides (number of pixel movement) over the image by computing its pixel values, with the maximum value calculated in that region and saved in the output image. Stride moves over all the pixels in the input image by taking high number from each stride and making a new image out of it. Below is the representation of this process.

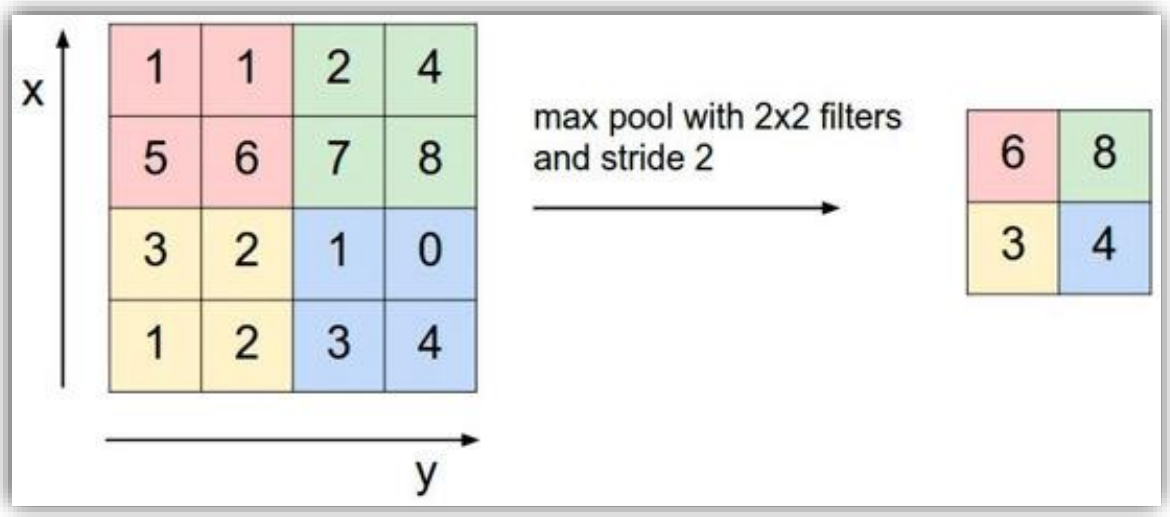


Figure 14. Example of max pooling

By max pooling we will be able to reduce the resolution of the convolutional layer, and its computational load in the network, additionally it also helps to reduce the overfitting in the model.

9.3 Data augmentation

It is the process of creating new data by making small modifications to old data. Because we're working with image data, the data augmentation process may include things like flipping the image horizontally or vertically, rotating the image, modifying the image's colour, and so on. The purpose of this

operation is to generate more data to feed into our model so that it can train more effectively. Some projects may have fewer images to train, resulting in overfitting owing to a lack of data, and so on. To overcome this problem with our model, we will use data augmentation. This technique will be better illustrated in the image below.

Figure 15. Example of data augmentation

9.4 Flatten layer

This is an important step that transforms the convolve image into a lengthy vector that can be fed into neural networks. The last classifier layer in a neural network is a dense layer, which expects the input to be a long vector. As a result, the output from the CNN's convolution layer or max pooling layer should be transformed into a one-dimensional feature vector that the neural network may use. This procedure is depicted in the image below.

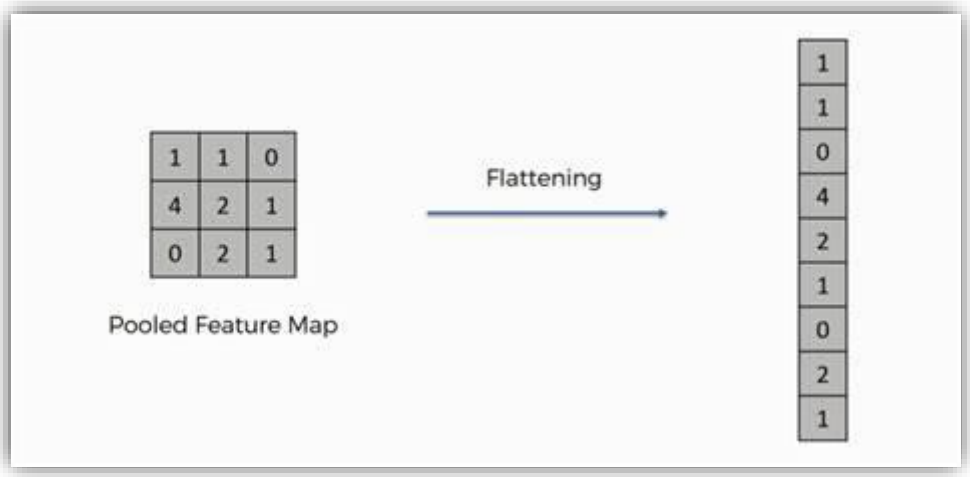


Figure 16. Example of flatten layer

9.5 Dropout

When we try to train our model with too much data it might overfit the model and perform poorly on the test dataset, dropout technique is the one of the regularization procedures which will tackle our overfitting problem and helps our model to perform better. When we try to run too many epochs in our models will overfit and perform poorly on test data set, any then just by dropping some random neurons in the network by giving the drop percentage, every time a input is given different neuron will get activated this in turn will reduce the overfitting problem.

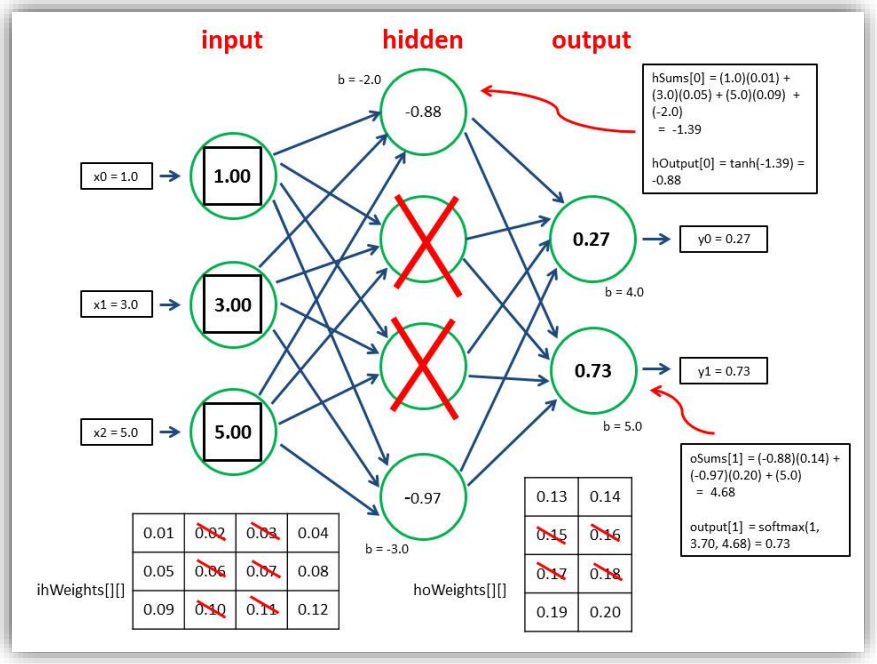


Figure 17.Dropout layer

9.6 Loss entropy

After the model classifies an output with a value indicating how relevant and accurate the prediction is, the loss entropy is calculated. This function will determine the error value for each input by comparing the output it has predicted with the actual value label for that input. In CNN, there are several loss entropies functions; the one we're utilising in this model is categorical cross entropy. If there are more than two classes, this is the type of loss entropy to use. SoftMax loss is a mix of SoftMax activation and cross entropy loss, and it's also known as SoftMax loss. The following equation represents this.

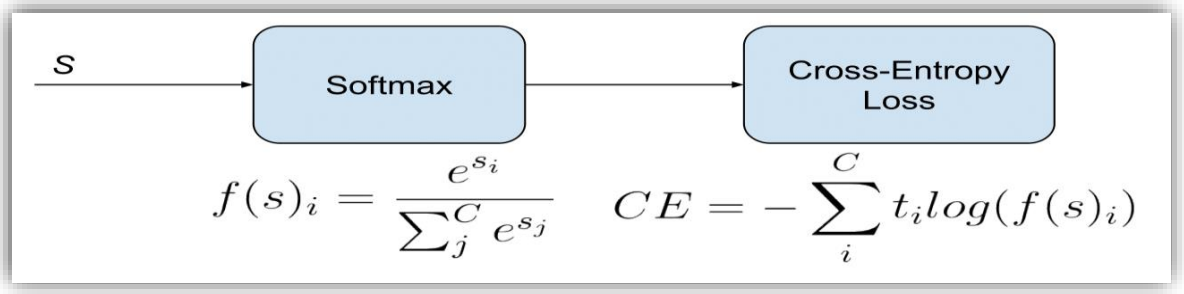


Figure 18. Loss entropy equation

9.7 Adam optimizer

By loss entropy we will be able to know our model performance, when our model starts to train with initial random weights our model might not classify correctly, then by back propagation our model will train by itself and updated the weights correctly by reaching the minimum loss as possible. The size of the step taken will be dependent on the learning rate so that the it won't pass the minimum and miss it. Adam optimizer will help us take these steps slowly initially and pick up speed overtime, and this is intuitively similar to the momentum, this way adam optimizer can take different size step for different parameters and with momentum with every parameter it can also lead to faster convergence, because of its speed and accuracy adam is used a default optimizer in many networks(Jais et al. 2019).

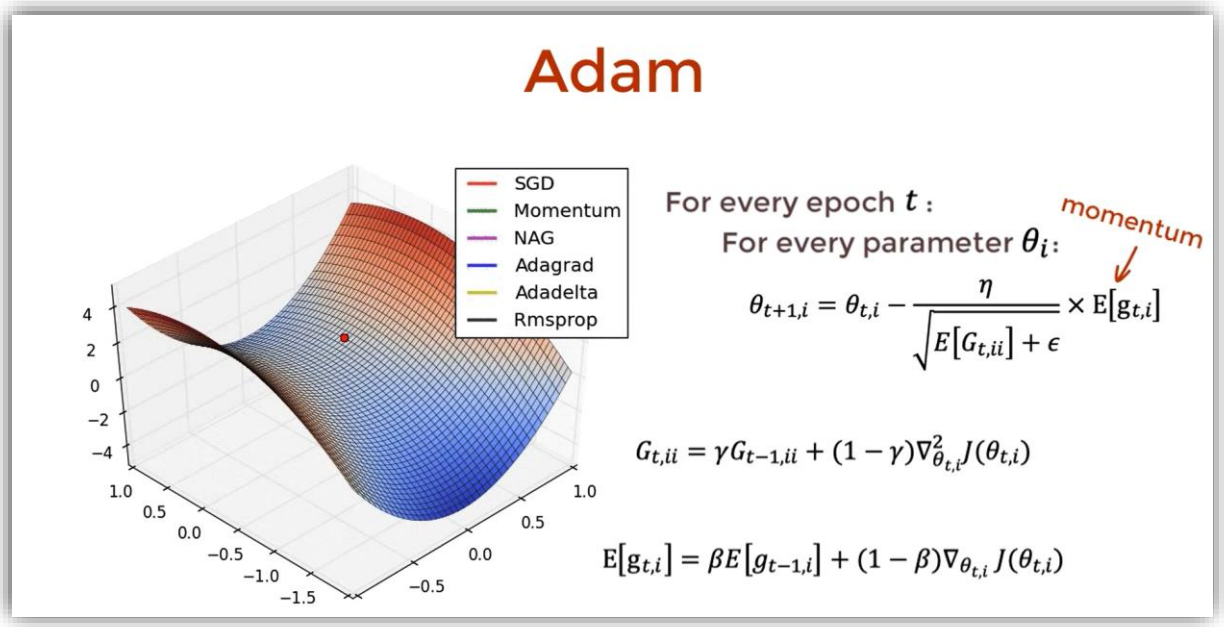


Figure 19. Adam optimizer

10. CNN model algorithm

a) Importing necessary Libraries.
b) Loading the dataset, assigning the data with 4 labels (Closed, Open, yawn, no yawn).
c) We are applying Haar cascade methods on image for ROI for 4 image classes.
d) Assigning X= features and Y= labels
e) Importing Label Binarizer function for labels
f) Train_test split data as Training = 70%, Testing=30%, Random state =42.
g) Applying image data generator to the data augmentation with following parameters, Rescale=1/255, zoom_range=0.2, horizontal_flip=True, rotation_range=30.
h) Building CNN model,
   1ST Layer – convolutional 256 layers and passed 3X3 kernel, activation function = ReLu
   Input image size as 145X145 and max padding = 2x2

   2nd Layer – Convolutional layer 128, 3x3 kernel, activation function= ReLu, max padding=(2x2)

   3rd Layer – Convolutional layer 64, 3x3 kernel, activation function= ReLu, max padding=(2x2)

   4th Layer – Convolutional layer 32, 3x3 kernel, activation function= ReLu, max padding=(2x2)

   5th Layer – Applying Flatten layer to bring it to one dimensional array with dropout = 0.5
   (if the value is less than 0.5 we are not including)

   6th Layer – we are passing dense layer with 64 layers and model dense 4 layer are output.
i) Saving the model data with their weights
j) We are testing new image with our model file.

11. Android application development

Artificial Intelligence has a wide range of applications in various industries, and it has the potential to take the world to the next level. The open-source platform has helped companies in a variety of industries use AI to reach more people and enhance their processes. Big firms have adopted these technologies, refined them, and made them available to the public so that they may easily solve real-world problems. For example, Google has established the ML kit APIs, a mobile SDK that will bring Google's on-device machine learning expertise to Android applications. These are very simple and powerful tools to use vison APIs to solve common problems and/or build new user experiences through applications. The ML Kit APIs are available on all platforms and allow you to create and share new ways to contribute to open source. Their capability is also available for devices that are not connected to the internet. This source is being used in this project to construct drowsiness applications utilising the Google API. The sections below explain how to use drowsiness application files.

11.1 Manifest file

This section is the main part of the application; it has all the metadata of this application which is represented as Android Manifest in this application. In this file we have stated with small activities like declaring the labels, icons, theme and etc, then we have moved on to call the google dependencies from online with a link. We've illustrated a VideoFaceDetectionActivity code that represents the application's principal action, such as the point at which it will start working.

11.2 Video Face Detection Activity

In this section we have created objects for camera preview, graphics overlay and camera sources followed by camera permission handler to allow this application to use mobile phone front camera. If runner satisfies if conditions, then will goes into Camera Source. Camera Source is a function which will open mobile camera and create one object for face detector. For this object we have setup a processor and created a class called Graphic Face Tracker Factory to use its inbuilt function called multiprocessor factory, for this function we have applied Graphic face tracker as override function.

11.3 Graphics Face Tracker

We have created this class for only two functions, firstly for graphics overlay and secondly for Face Graphics class in which graphics are seen on the mobiles screen is designed here with some algorithm which is explained in the next section.

11.4 Face Graphic

In this class we have assigned all the graphic overlay designs like colour of the boarder and style of text size display on the screen, front of the text etc, then comes the main algorithm to classify the drowsiness. Algorithm start with inbuilt method draw canvas and paint function to show graphic when eyes are closed along with the media sound, then goes on detecting face position and get the eye status with a probability between (0.4 to -1.0) to determine if they are closed or open. If the probability is within the ratio, then alarm initial will take place along with text canvas.

12. Workflow of the application

a) In android manifest application will start running by calling VideoFaceDetectionActivity
b) A processor is created to handle graphic face tracker class, by using multi-processor factor which is the in-built function to override graphic face tracker.
c) In the face graphics tracker processor, we have initialised overlay of the screen and called face graphic class.
d) Face graphic class is the main file in this application, in this class we have assigned all the graphic overlay designs, and called the algorithm,

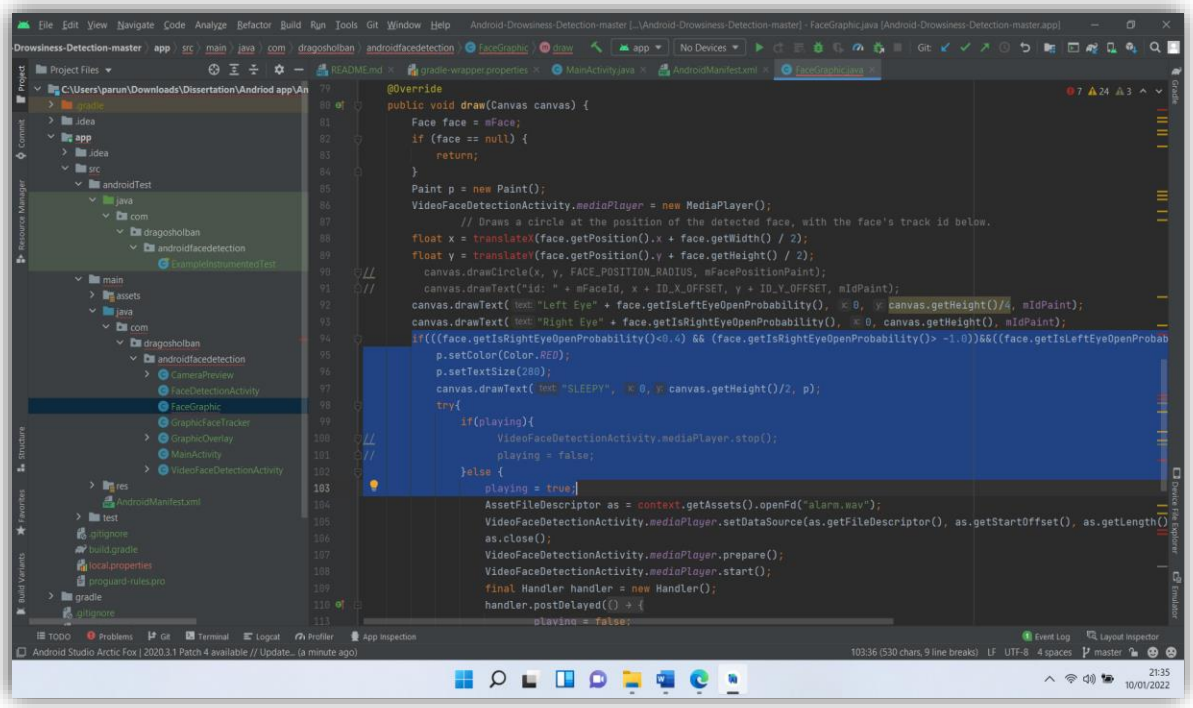


Figure 20. Screen shot of the Face Graphic section with algorithm

e) Eye ration probability is taken
f) If eye ration is between (0.4 to -1.0) alarm initial will take place along with text canvas.

13. Evaluation

In this section, I'll discuss the CNN model's results and drowsiness detection using a webcam, as well as the outcomes of the drowsiness model on a mobile application. We chose data with four labels for CNN model training: open eye, closed eye, yawning, and no yawning. And the CNN model's entire training and testing process is centred on the four characteristics. However, due to the covid pandemic, most individuals are compelled to wear face masks, making drowsiness detection impossible in a yawning scenario, thus we've simply added two criteria: open and closed eye.

Reproducible results

We know that each time we run the model, the updated weights may not be the same as the previous one, and this can affect prediction accuracy when testing the model with new data, so to get the same and best results every time, I used reproducible parameters by giving fixed random seed values of NumPy, Python, and TensorFlow for the Keras backend engine, and forcing TensorFlow to run in a single thread. Multiple threads have the ability to yield non-reproducible outcomes.

Below is the CNN model performance results.

Total learnable parameters in the CNN model we got as 495,140 as shown in the below figure,

```
=================================================================
 conv2d_1 (Conv2D)           (None, 143, 143, 256)     7168

 max_pooling2d (MaxPooling2D  (None, 71, 71, 256)      0
 )

 conv2d_2 (Conv2D)           (None, 69, 69, 128)       295040

 max_pooling2d_1 (MaxPooling  (None, 34, 34, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 32, 32, 64)        73792

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 14, 14, 32)        18464

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 flatten (Flatten)           (None, 1568)              0

 dropout (Dropout)           (None, 1568)              0

 dense (Dense)               (None, 64)                100416

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 495,140
Trainable params: 495,140
Non-trainable params: 0
_________________________________________________________________
```

Figure 21. Learning parameters of CNN

When we fit the models to the training dataset, we got the following results in the image: at the start, the loss function was 96 percent and the accuracy was 58 percent due to random weights assigned; after the first 10 epochs of training, the loss function was dramatically reduced and validation accuracy increased steadily; at the end of the 50th epoch, the training and validation accuracy were both 96 percent with a minimum loss function of 0.09%.

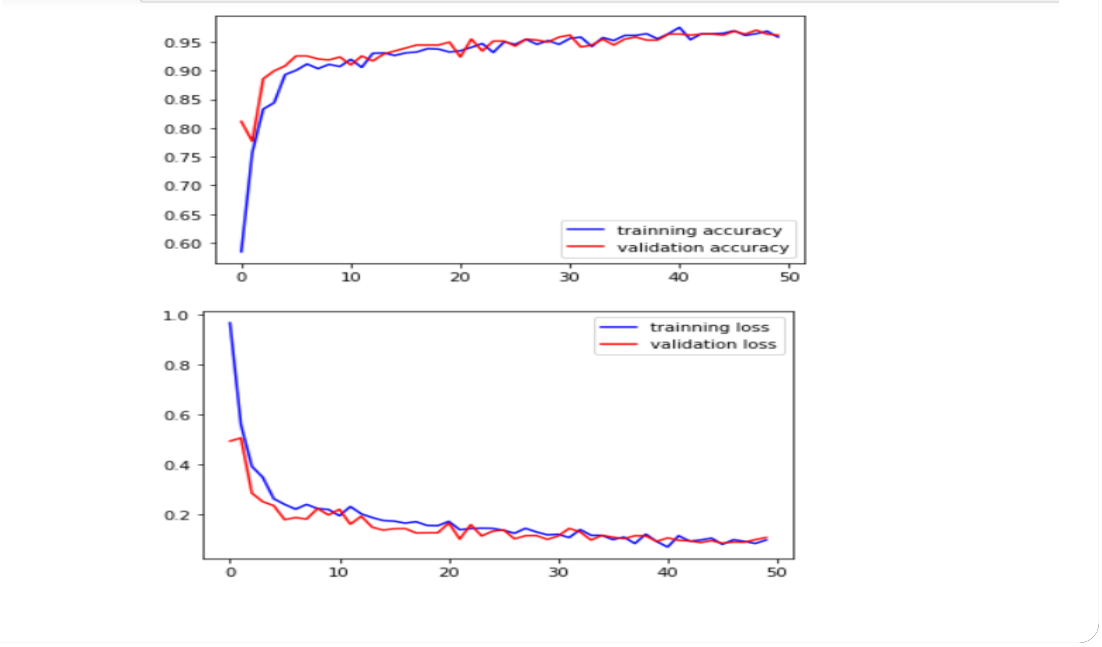


Figure 22. Training accuracy and loss

We observed that Open eyes had the best precision, whereas Closed eyes had the highest recall and f1-score in our test data set. Precision has a total of 98 percent true positive outcomes in open eyes and a recall rate of 99 percent true positive results in closed eyes. Test results are relatively close to training results, therefore we can say that our model is performing brilliantly in each of these cases.

	precision	recall	f1-score	support
yawn	0.82	0.78	0.80	63
no_yawn	0.79	0.84	0.82	74
Closed	0.88	0.99	0.93	215
Open	0.98	0.86	0.92	226
accuracy			0.90	578
macro avg	0.87	0.87	0.87	578
weighted avg	0.90	0.90	0.90	578

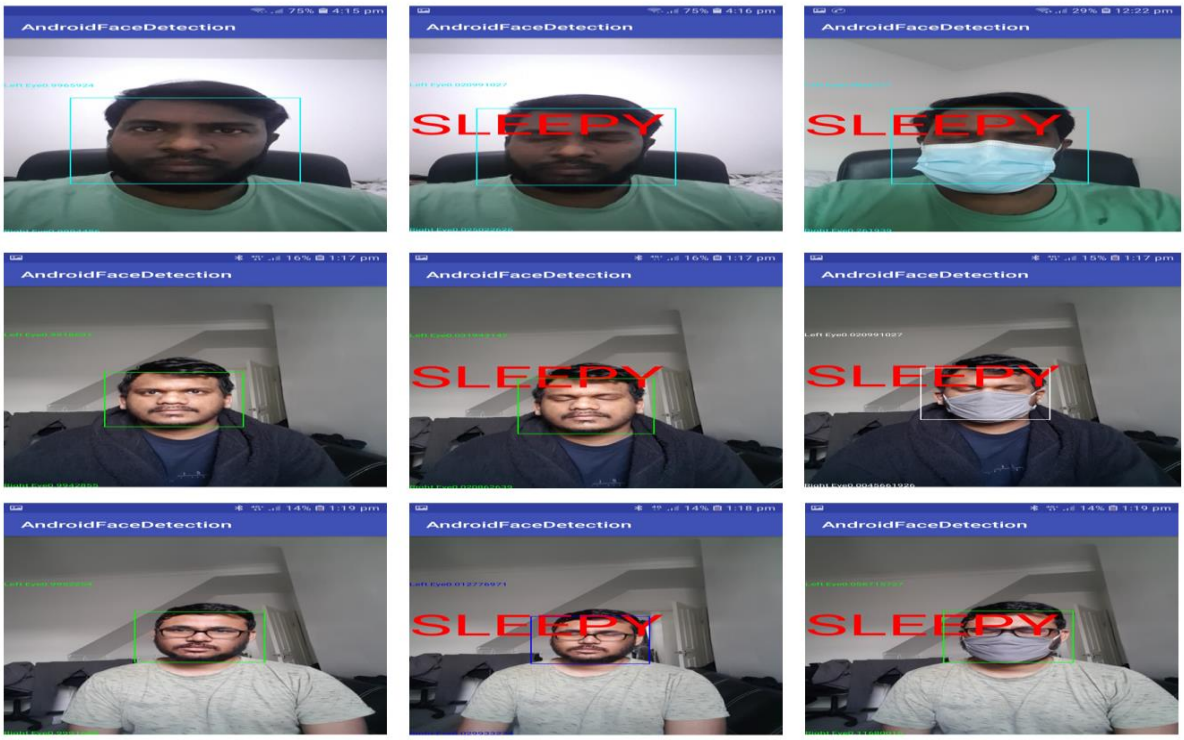
Figure 23. Model performance on test data

Drowsiness detection model results

We tried our model with several persons with masks on and off on a laptop and an Android application, and we obtained some good results. As stated, our model will sound an alarm if a person closes their eye for more than 5 seconds along with red borders on the window, as shown in the image below. Based on the alert message on image and count, we can say that our model properly predicted drowsiness.

Model testing using web cam

Model testing on Android application



14. Discussion and comparison:

In this section, we'll compare our model to the other models and outcomes discussed in the study paper I referenced at the start of this project. In addition, I will provide some intriguing observations in between the two studies.

In comparison to (Chirra, ReddyUyyala and KishoreKolli, 2019) both papers used the same method and obtained similar results; however, they were published in 2019 and did not compare their results to any other papers or make any further improvements to the model; however, their model was able to identify drowsiness with 96.42 percent accuracy and efficiently identify drowsiness.

In comparison to the technique they used, (Pattarapongsin et al., 2020) their strategy was centred on Pose estimate, Face landmarks, Eye Aspect Ratio, and Mouth Aspect Ratio. When compared to my model, this article has advanced and demanding elements; these are the crucial features that contribute to enhancing the drowsiness detection model, according to their paper, which was partially supported by the Centre of Excellence in Intelligent Informatics. There are many beneficial characteristics in this work; in the future, we may utilise transfer learning to adopt the learning from this model and test it in real time; if we are able to achieve good results, we will be able to eliminate all unusual situations, resulting in smart traffic analysis.

(Jabbar et al., 2018) was an improved version of the original paper in which they used a machine learning model to predict drowsiness, with the goal of reducing the size of the model so that it could be implemented in embedded systems such as Android applications while maintaining and achieving high performance. The concept of this paper, in my opinion, is the end concern of this project, as there are many new technologies readily available in the open-source platform and advancements in

hardware technology. However, this model still must be improved, as it only achieves 83.33 percent accuracy on average, compared to our paper ML kit, which was able to deliver APIs quickly and with little memory usage. Furthermore, our approach was able to yield some positive outcomes.

When a person is feeling drowsy, facial emotions and eye aspect ratio are two interrelated emotions (Ceamanunkul and Chawla, 2020). Most people do not express emotions when they are feeling drowsy, and there are other factors that may affect the performance of this model, such as sunglasses and masks, brightness, and so on. It is similar to our model when comparing factors that affect the model, but emotions can be ignored in this scenario.

This is one of the most intriguing papers I've ever read; (Sooksatra et al., 2018) proposes a method based on displacement and gradient vectors, which allows us to eliminate the light effect factor that causes models to perform poorly. Using this method, we can make our models independent of light intensity, which is a critical factor that causes any model to perform poorly, including face mask and pose estimation. We can learn about an essential topic from this study, which we can apply to our model in future advances.

15. Further developments

Due to the covid pandemic, most people are required to wear face masks; therefore, drowsiness detection when a person is wearing a mask is not possible in a yawning scenario. In the future, we can simply add a yawning scenario to the main drowsiness detection algorithm, and further improve with mask based on Jaw line landmarks, as we tend to open our mouth when we yawn. We can use the sensors in the steering wheel to monitor the driver's physical activity and incorporate this feature into our model for future improvements, and we can also deploy our model on a server and link to android applications so that data can be stored and used continuous improvement of the model.

16. Conclusion

In this research, we offer an android application and CPU-based drowsiness detection system based on Haar cascading files, CNN model, and ML kit APIs. The region of interest is retrieved using face detection and fed into the CNN model and ML kit APIs to determine whether a person is drowsy or not. Our suggested drowsiness detection model had good results in terms of training and validation, as well as user interface model implementation on both laptops and Android mobile devices. This was accomplished without the need of high-end technology or costly database administration. This approach was created using basic notions and has some restrictions that prevent it from working in all situations. We will aim to further improve the model and deploy it into a local server and connect it android APIs, so that user data can be saved, and the model may be re-trained continually for higher accuracy.

17. Legal Ethical and Professional Issues

This project's study data comes from Kaggle (yawn_eye_dataset_new | Kaggle), an open-source platform that provides data available for anyone to use, investigate, and contribute to society. This project does not involve any personal or third-party information, and it adheres to all of Teesside University's Research Ethics guidelines.

18. Reference

Chirra, V., ReddyUyyala, S. and KishoreKolli, V. (2019). Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State. Revue d'Intelligence Artificielle, 33(6), pp.461–466.

B, R.Sri., Y, Akanksha., R, Puthali. and T, Anuradha. (2021). Early Driver Drowsiness Detection using Convolution Neural Networks. 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC).

Biju, A. and Edison, A. (2020). Drowsy Driver Detection Using Two Stage Convolutional Neural Networks. 2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS).

Ceamanunkul, S. and Chawla, S. (2020). Drowsiness Detection using Facial Emotions and Eye Aspect Ratios. 2020 24th International Computer Science and Engineering Conference (ICSEC).

Celona, L., Mammana, L., Bianco, S. and Schettini, R. (2018). A Multi-Task CNN Framework for Driver Face Monitoring. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8576244 [Accessed 10 Jan. 2022].

Chirra, V., ReddyUyyala, S. and KishoreKolli, V. (2019). Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State. Revue d'Intelligence Artificielle, 33(6), pp.461–466.

freeCodeCamp.org (2020). Android Development for Beginners - Full Course. YouTube. Available at: https://www.youtube.com/watch?v=fis26HvvDII.

Goyal, M., Goyal, R. and Lall, B. (2020). Learning Activation Functions: A new paradigm for understanding Neural Networks. arXiv:1906.09529 [cs, stat]. [online] Available at: https://arxiv.org/abs/1906.09529.

Guo, G., Wang, H., Yan, Y., Zheng, J. and Li, B. (2020). A fast face detection method via convolutional neural network. Neurocomputing, 395, pp.128–137.

Jabbar, R., Al-Khalifa, K., Kharbeche, M., Alhajyaseen, W., Jafari, M. and Jiang, S. (2018). Real-time Driver Drowsiness Detection for Android Application Using Deep Neural Networks Techniques. Procedia Computer Science, [online] 130, pp.400–407. Available at: https://www.sciencedirect.com/science/article/pii/S1877050918304137.

Jais, I.K.M., Ismail, A.R. and Nisa, S.Q. (2019). Adam Optimization Algorithm for Wide and Deep Neural Network. Knowledge Engineering and Data Science, 2(1), p.41.

Kamilaris, A. and Prenafeta-Boldú, F.X. (2018). Deep learning in agriculture: A survey. Computers and Electronics in Agriculture, [online] 147, pp.70–90. Available at: https://arxiv.org/pdf/1807.11809 [Accessed 5 Aug. 2019].

Khushaba, R.N., Kodagoda, S., Liu, D. and Dissanayake, G. (2013). Muscle computer interfaces for driver distraction reduction. Computer Methods and Programs in Biomedicine, [online] 110(2), pp.137–149. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0169260712003008 [Accessed 10 Jan. 2022].

Khyat, J., Yasaswini, V., Reeshika, P., Roy, K., Kalpana, N. and Yamparala, R. (n.d.). Drowsiness Detection Using Machine Learning Algorithms. [online] Available at: http://junikhyatjournal.in/no_1_Online_21/54.pdf [Accessed 10 Jan. 2022].

Lu, X., Duan, X., Mao, X., Li, Y. and Zhang, X. (2017). Feature Extraction and Fusion Using Deep Convolutional Neural Networks for Face Detection. Mathematical Problems in Engineering, 2017, pp.1–9

Mansur, V. and Shambavi, K. (2021). Highway Drivers Drowsiness Detection System Model with R-Pi and CNN technique. 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT).

McDonald, A.D., Lee, J.D., Schwarz, C. and Brown, T.L. (2018). A contextual and temporal algorithm for driver drowsiness detection. Accident Analysis & Prevention, 113, pp.25–37.

Minichino, J. and Howse, J. (2015). Learning OpenCV 3 computer vision with Python : unleash the power of computer vision with Python using OpenCV. Birmingham, Uk: Packt Publishing, September.

Nguyen, D.-L., Putro, M.D. and Jo, K.-H. (2020). Eyes Status Detector Based on Light-weight Convolutional Neural Networks supporting for Drowsiness Detection System. IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society.

Patidar, P. (2019). Introduction to Keras — Deep Learning Library. [online] MLAIT. Available at: https://medium.com/mlait/introduction-to-keras-deep-learning-library-2844b39f0496#:~:text=Keras%20is%20a%20deep%20learning%20framework%20for%20Python.

Pattarapongsin, P., Neupane, B., Vorawan, J., Sutthikulsombat, H. and Horanont, T. (2020). Real-time Drowsiness and Distraction Detection using Computer Vision and Deep Learning. Proceedings of the 11th International Conference on Advances in Information Technology.

Pramod Singh and Avinash Manure (2020). Learn TensorFlow 2.0 : Implement Machine Learning and Deep Learning Models with Python. New York, Ny] Apress.

R, R., Goraya, A. and Singh, G. (2018). Real Time Drivers Drowsiness Detection and alert System by Measuring EAR. International Journal of Computer Applications, 181(25), pp.38–45.

Rajneesh, Goraya, A. and Singh, G. (2018). Real Time Drivers Drowsiness Detection and alert System by Measuring EAR. International Journal of Computer Applications, [online] 181(25), pp.38–45. Available at: https://www.ijcaonline.org/archives/volume181/number25/30095-2018918055 [Accessed 10 Jan. 2022].

Rimini-Doering, M., Altmueller, T., Ladstaetter, U. and Rossmeier, M. (2005). Effects of Lane Departure Warning on Drowsy Drivers' Performance and State in a Simulator. Driving Assesment Conference, [online] 3(2005). Available at: https://pubs.lib.uiowa.edu/driving/article/id/28213/ [Accessed 10 Jan. 2022].

Said, S., AlKork, S., Beyrouthy, T., Hassan, M., Abdellatif, O. and Abdraboo, M.F. (2018). Real Time Eye Tracking and Detection- A Driving Assistance System. Advances in Science, Technology and Engineering Systems Journal, 3(6), pp.446–454.

Shakeel, M.F., Bajwa, N.A., Anwaar, A.M., Sohail, A., Khan, A. and Haroon-ur-Rashid (2019). Detecting Driver Drowsiness in Real Time Through Deep Learning Based Object Detection. Advances in Computational Intelligence, pp.283–296.

Shakeel, M.F., Bajwa, N.A., Anwaar, A.M., Sohail, A., Khan, A. and Haroon-ur-Rashid (2019). Detecting Driver Drowsiness in Real Time Through Deep Learning Based Object Detection. Advances in Computational Intelligence, pp.283–296.

Sooksatra, S., Kondo, T., Bunnun, P. and Yoshitaka, A. (2018). A drowsiness detection method based on displacement and gradient vectors. Songklanakarin J. Sci. Technol, [online] 40(3), pp.602–608. Available at: https://rdo.psu.ac.th/sjstweb/journal/40-3/15.pdf.

Sooksatra, S., Kondo, T., Bunnun, P. and Yoshitaka, A. (2018). A drowsiness detection method based on displacement and gradient vectors. [online] www.semanticscholar.org. Available at: https://www.semanticscholar.org/paper/A-drowsiness-detection-method-based-on-displacement-Sooksatra-Kondo/a8eddc8e08de3125286e92606d0afb0994523935 [Accessed 10 Jan. 2022].

Tayab Khan, M., Anwar, H., Ullah, F., Ur Rehman, A., Ullah, R., Iqbal, A., Lee, B.-H. and Kwak, K.S. (2019). Smart Real-Time Video Surveillance Platform for Drowsiness Detection Based on Eyelid Closure. Wireless Communications and Mobile Computing, 2019, pp.1–9.

Tayab Khan, M., Anwar, H., Ullah, F., Ur Rehman, A., Ullah, R., Iqbal, A., Lee, B.-H. and Kwak, K.S. (2019). Smart Real-Time Video Surveillance Platform for Drowsiness Detection Based on Eyelid Closure. Wireless Communications and Mobile Computing, 2019, pp.1–9.

United States ed., (1998). PERCLOS, a valid psychophysiological measure of alertness as assessed by psychomotor vigilance. [online] Library Catalog (Blacklight). Washington, DC: Federal Highway Administration, Office of Motor Carriers, Office of Motor Carrier Research and Standards. Available at: https://searchworks.stanford.edu/view/9173359 [Accessed 10 Jan. 2022].

Viola, P. and Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. [online] Available at: http://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf [Accessed 10 Jan. 2022].

Wang, Q., Li, Y. and Liu, X. (2018). Analysis of Feature Fatigue EEG Signals Based on Wavelet Entropy. International Journal of Pattern Recognition and Artificial Intelligence, 32(08), p.1854023.

Weng, C.-H., Lai, Y.-H. and Lai, S.-H. (2017). Driver Drowsiness Detection via a Hierarchical Temporal Deep Belief Network. Computer Vision – ACCV 2016 Workshops, pp.117–133.

Weng, C.-H., Lai, Y.-H. and Lai, S.-H. (2017). Driver Drowsiness Detection via a Hierarchical Temporal Deep Belief Network. Computer Vision – ACCV 2016 Workshops, pp.117–133.

www.youtube.com. (n.d.). Complete Python NumPy Tutorial (Creating Arrays, Indexing, Math, Statistics, Reshaping). [online] Available at: https://www.youtube.com/watch?v=GB9ByFAIAH4 [Accessed 10 Jan. 2022].

Xie, Y., Chen, K. and Murphey, Y.L. (2018). Real-time and Robust Driver Yawning Detection with Deep Neural Networks. 2018 IEEE Symposium Series on Computational Intelligence (SSCI).

Xie, Y., Chen, K. and Murphey, Y.L. (2018). Real-time and Robust Driver Yawning Detection with Deep Neural Networks. 2018 IEEE Symposium Series on Computational Intelligence (SSCI).

Xue-Qin Huo, Zheng, W.-L. and Lu, B.-L. (2016). Driving fatigue detection with fusion of EEG and forehead EOG. 2016 International Joint Conference on Neural Networks (IJCNN).

Yu, C., Qin, X., Chen, Y., Wang, J. and Fan, C. (2019). DrowsyDet: A Mobile Application for Real-time Driver Drowsiness Detection. 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI).

Zhang, C., Wang, H. and Fu, R. (2014). Automated Detection of Driver Fatigue Based on Entropy and Complexity Measures. IEEE Transactions on Intelligent Transportation Systems, 15(1), pp.168–177.