California State University, Northridge

Detecting Drowsiness of Drivers using Machine Learning using inceptionv3 and DLIB

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

By

Kunj Shah

May 2023

The thesis of Kunj Shah is approved:

_____ _____

Robert McIlhenny, Ph.D., Chair Date

_____ _____

Katya Mkrtchyan, Ph.D. Date

_____ _____

Mahdi Ebrahimi, Ph.D. Date

California State University, Northridge

Acknowledgements

I am deeply grateful to Dr. Robert McIlhenny for his invaluable support and mentorship as my committee chair throughout my time at CSUN. His guidance and feedback have been instrumental in shaping my research and thesis. I also extend my heartfelt appreciation to Dr. Katya Mkrtchyan and Dr. Mahdi Ebrahimi for their valuable contributions as my committee members, providing me with valuable insights and suggestions that have enhanced the quality of my work.

I am also thankful to my parents for their unwavering love, encouragement, and support during my academic journey. Their constant belief in my abilities has been a driving force behind my accomplishments, and I am deeply grateful for their presence in my life.

Table of Contents

Signature Page ii

Acknowledgements iii

List of Figures vi

Abstract vii

1 Introduction 1

1.1 Background 1

1.2 Approach to Drowsiness Detection 2

2 Related Work 5

2.1 Overview of drowsiness detection 5

2.2 Previous studies on drowsiness detection 5

2.2.1 Image processing-based methods: 5

2.2.2 Yawning Based Technique 6

2.2.3 EEG based detection: 6

2.2.4 Eye Aspect Ratio 7

2.2.5 Machine learning based methods. 7

3 Technical Approach 10

3.1 Data Exploration: 10

3.2 Workflow to train a model. 12

3.3 General Workflow 14

3.4 Hardware 15

3.5 Software 16

4 Data Pre-processing 18

4.1 Getting Data 18

4.2 Understanding data's structure 18
4.3 Splitting data into two classes 20
4.4 Splitting data into training data and testing data 21
5 Models 23
5.1 Convolutional Neural Network (CNN) 23
5.2 Xception Model 23
5.3 InceptionV3 Model 26
5.4 Optimizer 28
6 Assessment Metrics 30
6.1 Confusion Matrix 30
6.2 Precision 31
6.3 Recall 32
6.4 Accuracy 33
6.5 F1 Score 34
6.6 Model Train Accuracy and Loss 34
6.7 Model Test Accuracy 35
6.8 Model Loss 35
7 Experiments 36
8 Results 41
9 Conclusion and Future Work 44
References 45

List of Figures

Figure 1 - Face Detection in Images (DATATURKS)..... 11

Figure 2 - Open and Close eyes (MRL Eye Dataset)..... 12

Figure 3 – Workflow..... 14

Figure 4 – General workflow of the system..... 15

Figure 5 – Steps for Data preprocessing..... 18

Figure 6 – Annotated Images of the proposed dataset..... 19

Figure 7 – Open Eye / Close Eye Data split…….…….…….……. 20

Figure 8 – Train / Test Data split……...…….....…… 22

Figure 9 – Xception Model Brief Architecture……...…….…….... 24

Figure 10 – Inception v3…….……..……26

Figure 11 – Confusion Matrix…….…….…....……. 30

Figure 12 – Xception Model Training Batch size 32 1% Testing Data…….……..……. 36

Figure 13. Xception Model Accuracy and Loss Batch Size 32 1% Testing Data…….37

Figure 14. Xception Model Accuracy and Loss 1% Testing Data tweaked..……37

Figure 15. Xception Model Accuracy and Loss Batch Size 16 10% Testing Data..…….38

Figure 16. Xception Model Accuracy and Loss 10% Testing Data Tweaked.…….….…39

Figure 17. InceptionV3 Model Accuracy and Loss Batch Size 16 10% Testing Data……39

Figure 18. InceptionV3 Model Accuracy and Loss 10% Testing Data Tweaked……...…….40

Figure 19. Confusion Metrics for Xception…….……..……41

Figure 20. Confusion Metrics for Inceptionv3…….…….…….…….…….…….....…….43

Abstract

Detecting Drowsiness of Drivers using Machine Learning using inceptionv3 and DLIB

By

Kunj Shah

Master of Science in Computer Science

Around the world, traffic accidents and fatalities are frequently caused by drowsy drivers. The National Highway Traffic Safety Administration (NHTSA) estimates that drowsy driving causes 100,000 car accidents, 71,000 injuries, and 1,500 fatalities every year in the United States alone. These numbers show how seriously drowsy driving affects American road safety. In the US, boring highway and interstate conditions and extended driving hours can also make it more likely that a motorist will become fatigued.

This thesis uses the machine learning techniques dlib and inceptionv3 to suggest a novel approach for detecting driver drowsiness. The suggested solution makes use of dlib for detecting face landmarks and a convolutional neural network (CNN) based on the inceptionv3 architecture for feature extraction. The Media Research Lab Eye Dataset and a custom dataset of facial images are used to train the CNN model. A Support Vector Machine (SVM) classifier uses the retrieved features to predict the driver's level of tiredness. With an accuracy of 96.5% for determining drowsiness results from experiments on the real-world dataset show the efficiency of the suggested strategy. The proposed approach provides a dependable solution for real-time

driver sleepiness detection, which has the potential to improve traffic safety and avoid accidents caused by driver fatigue.

Therefore, detecting driver drowsiness can help in reducing the chance of crashes caused by driver weariness, thereby saving lives and reducing the number of traffic accidents in the US.

1 Introduction

1.1 Background

Around the world, traffic accidents and fatalities are frequently caused by drivers' drowsiness. The estimates given by National Highway Traffic Safety Administration (NHTSA) is drowsy driving causes 100,000 car accidents, 71,000 injuries, and 1,500 fatalities every year in the United States alone. Driving when sleepy can affect a person's physical as well as mental faculties, which leads to longer reaction times, decreased focus, and poorer judgment. For the sake of road safety and to minimize accidents caused by driver weariness, it is essential to spot drowsiness in drivers.

Traditional methods to determine driver weariness include self-reporting, behavioral observations, and physiological tests like electroencephalography and heart rate. (EEG). These approaches have limitations, however, like being obtrusive or requiring equipment's. Convolutional neural networks (CNNs), for example, have been suggested as a non-intrusive yet effective way to detect driver drowsiness.

Modern performance on image classification problems has been achieved by the inceptionv3 architecture, a CNN model. Facial landmark detection, face detection, and other image processing operations are a part of the popular computer vision library known as DLIB. It is possible to create a novel method for detecting driver drowsiness using machine learning by combining these methods.

In this thesis, we provide a method to detect driver drowsiness using the machine learning techniques DLIB and Inceptionv3. The proposed approach combines DLIB for facial landmark identification and a CNN built on the Inceptionv3 architecture for feature extraction. A dataset of facial images taken from drivers and MRL eye dataset is used to train the CNN model. A Support Vector Machine (SVM) classifier uses the features obtained to predict the driver's level of drowsiness The proposed approach offers a trustworthy and unobtrusive alternative for real-time driver sleepiness identification, which has the potential to improve traffic safety and avoid accidents brought on by driver exhaustion.

1.2 Approach to Drowsiness Detection

Drowsiness detection analysis encompasses various approaches, which can be employed based on factors such as the specific problem statement, available time and resources, and desired accuracy levels. The choice of method may vary depending on the unique requirements of the project.

1.2.1. Eye Aspect Ratio Based: Drowsiness detection in computer vision applications frequently employs the Eye Aspect Ratio (EAR) method. The degree of eye closure can be estimated using the EAR, which is a measurement of the ratio between the length and width of the eye region in an image.

Here is a general approach for drowsiness detection using EAR:

Detect the eyes in the input image using an object detection algorithm.

Extract the eye regions from the input image.

Calculate the EAR for each eye using the following formula:

$$EAR = (|p2-p6| + |p3-p5|) / (2 * |p1-p4|)$$

where p1, p2, p3, p4, p5, and p6 are the coordinates of the six points that define the eye region in the image.

Calculate the average EAR for both eyes.

If the average EAR is below a certain threshold, it indicates that the eyes are partially or fully closed, which may be a sign of drowsiness.

Trigger an alarm or alert to notify the driver or user that they are exhibiting signs of drowsiness.

1.2.2. Machine Learning Based:

In machine learning-based drowsiness detection, different variables, including eye movements and facial expressions are analyzed by algorithms to determine whether an individual is drowsy or weary. This method often involves collecting sensor data and using it to build a machine learning model that may later be applied to real-time drowsiness detection. Deep learning, support vector machines (SVMs), and decision trees are a few typical machine learning methods utilized in sleepiness detection. Drowsiness can be a risk factor in a variety of situations, including driving, transportation, and other professions. Overall, machine learning-based drowsiness detection has the potential to increase safety in these scenarios.

1.2.3. Electroencephalography (EEG): Electroencephalography (EEG) technology is used in the EEG signals approach to identify variations in patterns of brainwaves that signify drowsiness. By placing electrodes on the scalp, the EEG is a non-invasive technique for determining the electrical activity of the brain. These devices track changes in the brain's electrical activity, which can be evaluated to gauge how sleepy the driver is.

Studies have demonstrated that tiredness in drivers may be detected using EEG signals. Frequency analysis of the EEG signals is a popular technique for spotting changes in brainwave patterns.

There are some other methods as well such as Heart Rate Monitors and Steering Wheel Movement Sensors. After examining all the options and comprehending the modifications and requirements associated with each one, we believe the overhead associated with these ways is excessive. We propose a system that avoids overly complex modification or significantly altering how a person drives in order to meet this challenging criterion to implement Detection of Drowsiness. According to the suggested technique, drivers would only need to connect one module near their rearview mirror, from which point the camera could clearly see the driver's face. This approach merely entails the integration of a module with a device like the Raspberry Pi that is running a program that processes data collected by a camera before feeding it to a machine learning model that predicts each frame and calculates a score based on how long the driver's eyes were closed.

2 Related Work

2.1 Overview of drowsiness detection

Drowsiness detection is the process of determining exactly when someone is feeling sleepy or drowsy. Detecting sleepiness when driving is crucial since it can have a major impact on a person's capacity to drive safely. Drowsiness detection techniques come in a variety of forms, and they can be broadly divided into three categories: behavioral, physiological, and technical.

Different sensors or gadgets, such as eye-tracking technologies, facial landmark detection, and machine learning algorithms, are used in technological ways to detect drowsiness. Due to the possibility of integrating these methods into driver assist systems, they can be non-intrusive and quite precise.

Overall, real-time drowsiness detection has grown increasingly accessible because of technological improvements, which is essential for ensuring driver safety. However, the implementation and context in which it is utilized may determine the best strategy for drowsiness detection.

2.2 Previous studies on drowsiness detection

2.2.1 Image processing-based methods:

Lee and Lee (2018) [2] proposed an efficient and non-intrusive system for monitoring driver fatigue caused by fatigue, which is a major cause of traffic accidents. The proposed system uses support vector machine (SVM) for face extraction and circular Hough transform (CHT) for mouth detection. The system does not require any training data or special cameras and can be applied to real video sequences acquired by low-cost web cameras under various lighting

conditions. The experimental results show the performance of the system in detecting yawning and hence driver fatigue.

2.2.2 Yawning Based Technique

Driver's Fatigue Detection Based on Yawning Extraction [2]. Fatigue is a significant cause of traffic accidents, which endangers the safety of drivers and passengers. In response to this issue, a nonintrusive system for monitoring driver fatigue through yawning detection is proposed in this paper. The system utilizes a support vector machine (SVM) with face extraction and a novel mouth detection technique based on circular Hough transform (CHT) for yawning extraction. No training data or special cameras are required for the proposed system. The effectiveness of the system is demonstrated through experiments conducted on real video sequences captured by low-cost web cameras under different lighting conditions. (Gholami et al., 2014)

2.2.3 EEG based detection:

The research paper focuses on the detection of driver drowsiness using EEG power spectrum analysis [6]. The paper highlights the significance of addressing the critical issue of driver drowsiness, which can lead to fatal accidents, injuries, and property damages. It emphasizes that extensive research has been conducted in this area to tackle this problem effectively. The study utilized a driving simulator-based approach to investigate the changes in the EEG power spectrum during monotonous driving. The participants in the study comprised of nine healthy university students. The researchers computed the absolute band power of the EEG signal using the Fast Fourier Transform (FFT) technique, and the power spectral density was computed using the Welch method. The study found that alpha and theta band powers

significantly increased (p<;0.05) when a subject moved from an alert state to a drowsy state. When compared to other regions, these changes were more noticeable in the occipital and parietal regions. The results of the study show a promising drowsiness indicator that might be utilized for avoiding road accidents caused by driver drowsiness.

2.2.4 Eye Aspect Ratio

The study describes a real-time drowsiness detection system for drivers based on the Eye Aspect Ratio and Eye Closure Ratio. The device aims to identify driver drowsiness in the moment, as it is an essential problem that can lead to accidents and compromise road safety. The EAR and ECR are calculated using real-time study of eye characteristics, and the device is designed to provide early warnings or alerts for preventing drowsiness. [5]. The report highlights drowsy driving's significant impact as a primary cause of road accidents and fatalities on a global scale. Conventional methods for detecting drowsiness are frequently obtrusive, distracting, or require the use of expensive equipment. As a result, the study presents a real-time and lightweight drowsiness detection system constructed as an Android app. To identify drowsiness the system records videos and uses image processing algorithms to detect the driver's face, followed by computing Eye Aspect Ratio and Eye Closure Ratio using adaptive thresholding. The approach combines machine learning algorithms and achieves an accuracy of 84% using the random forest classifier.

2.2.5 Machine learning based methods.

The paper presents a novel approach for real-time detection of driver drowsiness using deep learning and object detection techniques [8]. The proposed methodology employs Convolutional Neural Networks (CNN) to effectively identify signs of driver drowsiness, aiming to prevent vehicle accidents caused by drowsy drivers. In the proposed methodology, drowsiness

detection is approached as an object detection task, aiming to detect and localize open and closed eyes from a real-time video stream captured from a driver. The methodology treats the task of drowsiness detection as a visual object recognition problem, where the system is designed to identify the presence and position of open and closed eyes in the video stream. In this task, the MobileNet Convolutional Neural Network (CNN) architecture is combined with the Single Shot Multi box Detector (SSD) to detect driver drowsiness. The output from the MobileNet-SSD architecture is then fed into a separate algorithm for driver drowsiness detection. To train the model, the authors curated a custom dataset of approximately 6,000 images that were labeled with objects such as face, eye open, and eye closed. They achieved a Mean Average Precision (mAP) of 0.84 on these categories using the PASCAL VOC metric. One of the notable advantages of this methodology is its computational efficiency and cost-effectiveness, as it allows real-time processing of incoming video streams on standalone mobile devices without the need for expensive hardware support. This makes it suitable for deployment on inexpensive embedded devices in vehicles, such as the Raspberry Pi version 3 or a mobile smartphone.

2.2.6 Deep Learning Based Approach

Deep Learning Based Approach for Real-time Driver Drowsiness Detection [9] The research presents a Convolutional Neural Network (CNN) model that aims to detect driver drowsiness in real-time, addressing the critical issue of drowsy driving and its impact on reducing accidents. The proposed model utilizes the Center Face algorithm for face detection and Haar-Classifier for eye feature extraction. It demonstrates accurate detection of driver drowsiness or alertness, even in challenging conditions such as the presence of glasses or masks. The model achieves impressive training and validation accuracies of 97% and 98%, respectively, with test accuracy of 97% for non-glassed eye datasets and 92% for glassed eye datasets. Notably, the

proposed approach performs well in real-time environments, showcasing its potential for practical implementation.

3 Technical Approach

3.1 Data Exploration:

When starting to explore and get more insight on this topic and after reading multiple sources, I was certain that going through multiple sources is necessary to decide what to proceed with.

Some of the datasets which I felt were relevant to this task are:

3.1.1. Face Detection in Images [10]: Image bounding box dataset to detect faces in images. This dataset had images with bounding boxes around the faces in them. 500 or so images with 1100 or so and they were manually labelled faces using bounding boxes. The images quality was pretty good. But after considering how little were the images and faces in those images, I was not sure if I should use it dataset as I would need more images.

3.1.2. Human Faces [11]: A dataset of human faces was collected through web scraping, which was then recommended for use in image processing models. 7.2k+ photos in a collection with numerous uses, including image identifiers and classification techniques. With some GAN-generated photos included as well to help with the functionality of identifying between genuine and produced faces, a thorough mixture of all common creeds, races, age groups, and profiles is used to try to create a neutral dataset. The dataset includes images of both male and female individuals, with different ages, ethnicities, and facial expressions. The images are labeled as either "male" or "female", and there are approximately equal numbers of each gender. The above dataset has plenty images and has human faces which overcame the problem faced in the Face Detection in Images dataset [10]. I downloaded the dataset and after analyzing this dataset the major issue was

to find images with closed eyes. There were images of individuals which did have eyes closed but they were not enough to train a model on [Open eyes, Close eyes] classes.

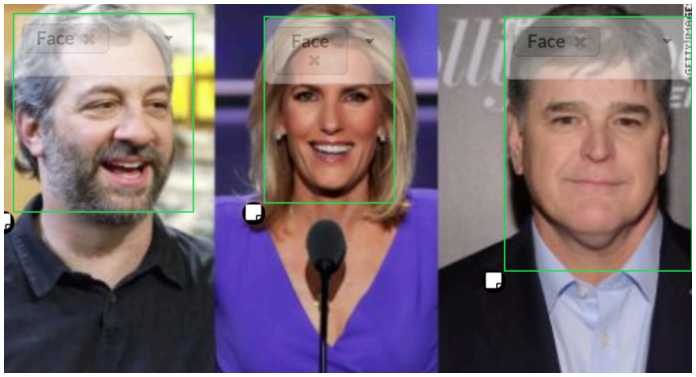

Figure 1. Face Detection in Images (DATATURKS)

3.1.3. Media Research Lab Eye Dataset [12]: Significant tasks in the field of computer vision encompass the recognition of eyeballs and their components, estimation of gaze direction, and measurement of eye-blinking frequency. We have been working on these tasks involving driver over the course of recent years, extensive data collection efforts have been carried out, resulting in a substantial amount of testing data being accumulated under actual circumstances. As a result, we present the MRL Eye Dataset is a comprehensive collection of infrared photos of the human eye that includes both low- and high-resolution images. The photos were acquired in a variety of lighting settings and with a variety of equipment, making them appropriate for testing various features and trainable classifiers. The dataset is structured into numerous categories, making it perfect for constructing and determining classifiers and allowing for comparison of alternative techniques. This dataset looks best so far, as the number of images is large in number as well as it contains data of closed and open eyes which we can use to train the model to classify images. Basically, this dataset overcomes disadvantages of discussed above each dataset.

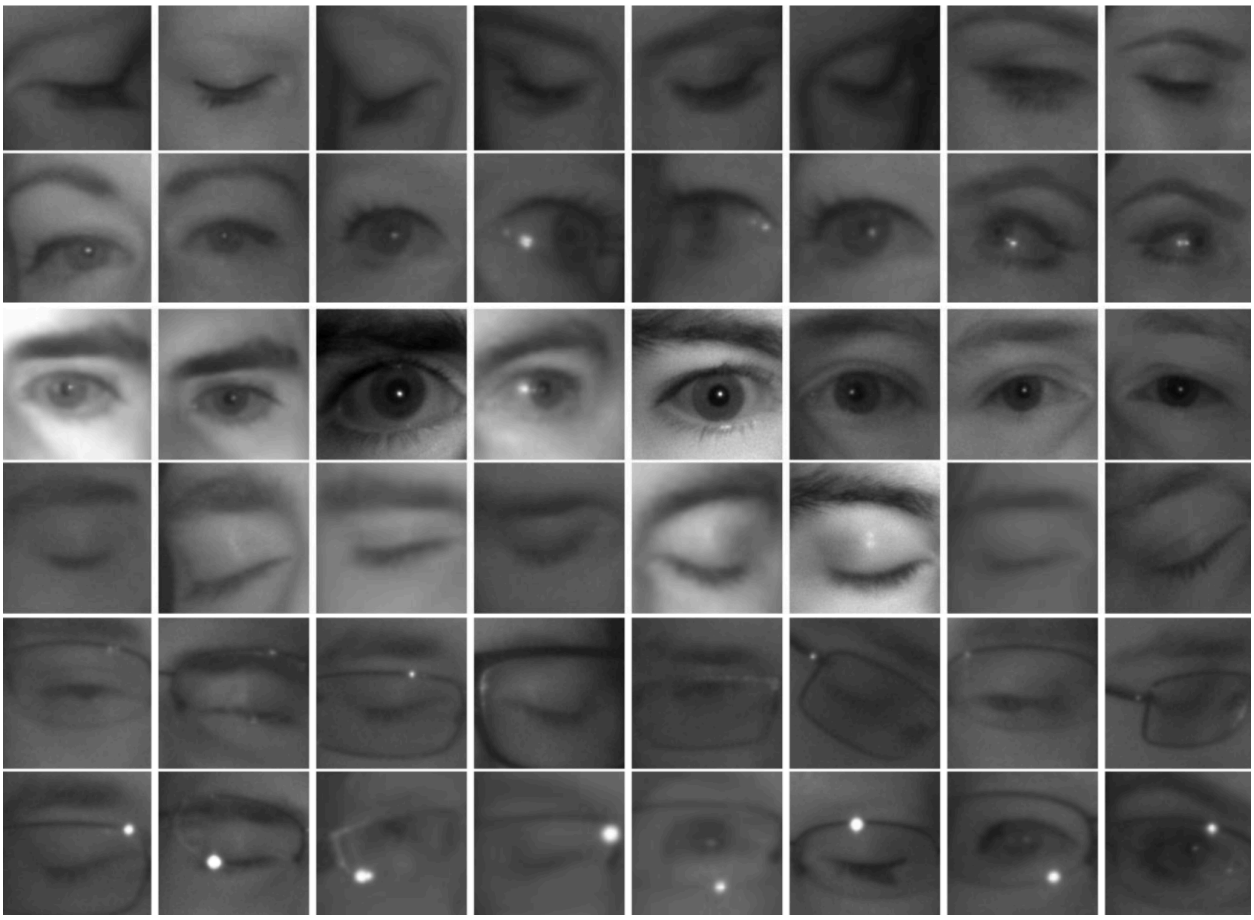

Figure 2. Open and Close eyes (MRL Eye Dataset)

After going through different datasets, we deiced to move forward with the MRL Eye Dataset as if serves the purpose we are looking to address, although the purpose of this dataset was different, but the similarity between the dataset we would need, and this dataset is high and is fulfilling all our needs.

3.2 Workflow to train a model.

A workflow for drowsiness detection can be divided into several steps. Following includes brief about each step involved in the process of building a system for Detecting Drowsiness in a driver while driving.

- Data preprocessing: Remove any outliers, noise, or artifacts from the collected data. This step also involves extracting and rearranging data as required.
- Feature Extraction: Take the preprocessed data and extract useful features like blink frequency, blink duration, eye closure time, and yawning frequency.
- Feature Selection: The most relevant features from the extracted feature set using techniques such as Principal Component Analysis (PCA), Recursive Feature Elimination (RFE), or other statistical or machine learning-based approaches.
- Model Development: Create a model using the chosen features that may identify drowsiness for this, various machine learning techniques can be used, such as Support Vector Machines (SVM), Neural Networks (NN), Random Forest (RF), and Logistic Regression.
- Model training: Use labelled data that covers both drowsy and non-drowsy states, or open eyes and closed eyes, to train the model. The labelled data may be gathered by naturalistic observation, such as driving in real life, or in controlled situations, such as a driving simulator.
- Model Evaluation: Determine the model's performance on a different testing set which consists of both sleepy and awake states. The effectiveness of the model can be assessed using various performance metrics, such as accuracy, precision, recall, and F1-score.
- Deployment: The model can be used in practical scenarios. Identify drowsiness and alert the driver or operator after it has been validated and put to the test.

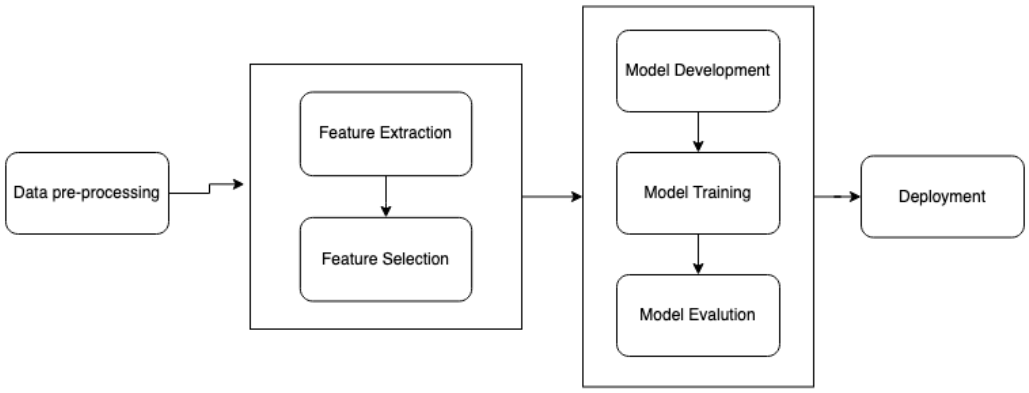

Figure 3. Workflow

3.3 General Workflow

This workflow is to understand how the system will work after we have the model trained and when we are integrating the model with camera, Raspberry Pi 4 and our program.

- Import the necessary libraries such as OpenCV, Dlib, Tensorflow, etc.
- Load the facial landmark detector and the trained CNN model.
- Initialize the video stream and start capturing frames.
- For every frame, utilize the Dlib library to identify and track facial landmarks.
- Extract the relevant features from the facial landmarks, such as eye closure duration or eye aspect ratio.
- Feed the extracted features into the trained CNN model to identify the level of drowsiness.
- Display the results on the screen or trigger an alarm if the person is becoming drowsy.
- Continue capturing frames until the user stops the program.

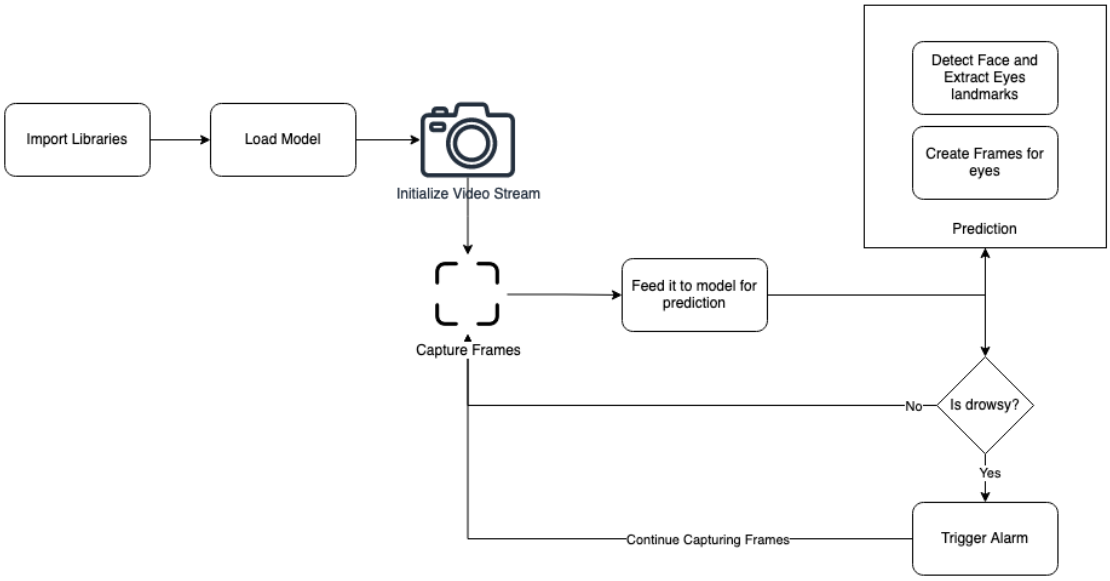

Figure 4. General workflow of the system

3.4 Hardware

- Camera: A camera is required to record the driver's facial expressions and eye movements in real-time. A high-resolution camera with low light sensitivity is recommended for better accuracy.
- Processor: The processor is the of the system, which performs all the computations required for drowsiness detection. Raspberry Pi 4 should be an option for our system as we will upload to the board.
- Memory: Sufficient RAM is required to store the collected data and run the machine learning models. A minimum of 8 GB of RAM is recommended for smooth operation. Raspberry Pi 4 does facilitate 8 GB RAM.
- GPU: A dedicated GPU can significantly increase the machine learning computations. A high-performance GPU such as NVIDIA GeForce GTX 1080 or equivalent is recommended for deep learning models like Inceptionv3. This device is only necessary

while training the model. Having a GPU is beneficial in deployment too for smoother functioning, but we can use a GPU which works with Raspberry Pi 4 as we do not need too heavy GPU for using the model and predicting the state of drowsiness of a driver. Possible attachments for Raspberry pi 4 are Radeon HD 5000/6000/7000.

- Power Supply: The system must have a reliable power supply to avoid any interruptions during operation. As this system is going to be added to the car's dashboard or around the rear mirror, also our systems are needed once a person state driving, we can draw power from the car as the consumption of Raspberry Pi 4 is not significant when compared to a car.

3.5 Software

Below are some of the possible technological details we might have to consider before deciding all the necessary software required for our system.

- Operating System: OS is must as our program will run on operating system. The system should have a supported operating system like Windows, macOS, or Linux. Preferred is Linux as there are certain flavors of OS which are light weight and are completable and convenient to be used on Raspberry Pi 4.
- Python: Python is the primary language used for machine learning and computer vision tasks. The latest version of Python 3.x should be installed.
- Other required packages: Other required packages include numpy, pandas, scikit-learn, and matplotlib for data manipulation, machine learning, and data visualization. Imutils is a library that provides convenient functions for performing common image processing tasks such as rotation, resizing, translation, displaying Matplotlib images, skeletonization, sorting contours, detecting edges, and more, making these operations easier to implement

with OpenCV. TensorFlow: TensorFlow is a popular deep learning framework used for developing and training machine learning models. Install the most recent version of TensorFlow to implement the Inceptionv3 model. Dlib: Dlib is a modern C++ toolkit that incorporates machine learning techniques as well as tools for constructing complex software applications. Install the latest version of Dlib to use its face detection and facial landmark detection capabilities.

The table provided below outlines the languages, tools, and frameworks that have been utilized during my experiments.

Operating System	Linux
Language	Python 3
Environment	Anaconda
IDE	Jupyter Notebook, VSCODE
Libraries	DLIB, Pandas, Imutils, Sci-Kit learn, Pygame, Keras, Numpy, TensorFlow, OpenCV, MatPlotLib.

Table 1 - Software specifications

4 Data Pre-processing

Data Pre-processing tasks

In this section we have explained the steps required for pre-processing the images dataset which we selected in Data Exploration.

These steps include getting the data, understanding the structure of the data, understanding the annotation used for naming files, separating data into classes and lastly splitting data into train and test.

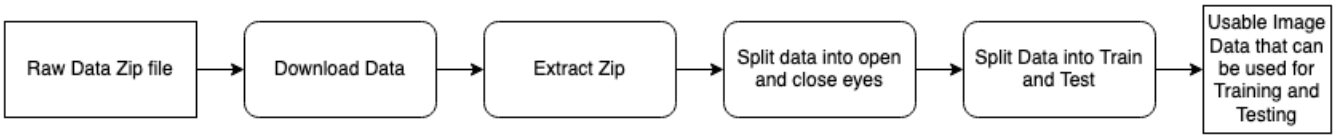

Figure 5. Steps for Data preprocessing

4.1 Getting Data

MRL Eye Dataset, this large-scale dataset of human eye images compressed file can be downloaded from [12] their page.

4.2 Understanding data's structure

Below is the description and structure of the dataset explained for better understanding the processing we need to use this dataset.

The following properties (which are listed below) were annotated in the dataset [12]:

- image ID: the dataset consists of 84,898 images.

- Subject ID: in the dataset, they collected the data of 37 distinct individuals (out of which 33 where men's and 4 women's).
- Gender: [0 - man, 1 woman] The dataset includes gender information for each image, categorizing them as male or female.
- Glasses: [0 - no, 1 - yes]: The presence or absence of eyeglasses in each eye image is indicated in the provided information.
- Eye state: [0 - closed, 1 - open] This characteristic applies to the open and closed states of the eyes.
- Reflections [0 - none, 1 - small, 2 - big]; They categorized the reflection states into three groups based on the size of reflections, namely no reflection, small reflections, and large reflections.
- Lighting conditions [0 - bad, 1 - good]; Each image is categorized into two states, either "bad" or "good", based on the amount of light present during the video capture.
- Sensor ID [01 - RealSense, 02 - IDS, 03 - Aptina]; The dataset includes images captured by three distinct sensors. (Intel RealSense RS 300 sensor with 640 x 480 resolution, IDS Imaging sensor with 1280 x 1024 resolution, and Aptina sensor with 752 x 480 resolution)

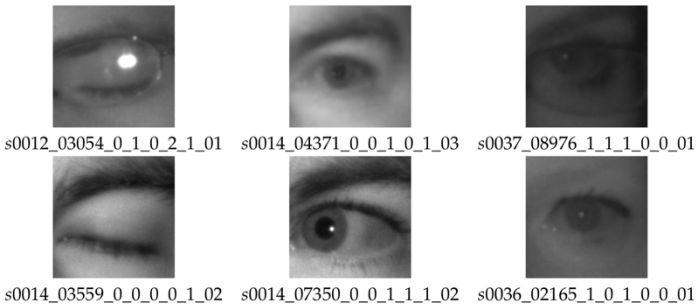

Figure 6. Annotated Images of the proposed dataset

4.3 Splitting data into two classes

The MRL Eye dataset webpage describes the structure of data. The data is collected by 37 individuals. After unzipping this 326MB zip file, we get 37 folders and annotation text file. Each folder has images with either close or open eyes. As the data is annotated, by the mentioned structure, we will have to parse each folder and each image inside each folder to check the file name to understand if the eye is open or closed. Below is the attached code snippet for dividing this image into two folders, open eyes, and close eyes.

```
Raw_DIR= '../mrlEyes_2018_01'
for dirpath, dirname, filenames in os.walk(Raw_DIR):
    for i in tqdm([f for f in filenames if f.endswith('.png')]):
        image = cv2.imread(dirpath+'/'+i)
        if i.split('_')[4]=='0':
            shutil.copy(src=dirpath+'/'+i, dst='prepared_data/close_eyes')

        elif i.split('_')[4]=='1':
            shutil.copy(src=dirpath+'/'+i, dst='prepared_data/open_eyes')
```

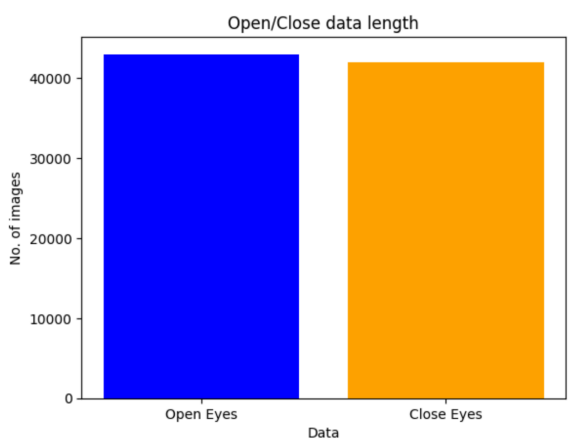

Figure 7. Open Eye / Close Eye Data split

4.4 Splitting data into training data and testing data

After splitting the data into open eyes and close eyes, we would also split the data into train and test. We would also have to split the train and test data into train-close-eyes, train-open-eyes, test-close-eyes, and test-open-eyes.

To move 415 close eye images to test folder.

```
parent_folder = "prepared_data/close_eyes"
files = os.listdir('prepared_data/close_eyes')
for each in range(len(files[:4015])):
  old_path = f"{parent_folder}/{files[each]}"
  new_path = f"prepared_data/test/close_eyes/{files[each]}"
  shutil.move(old_path, new_path)
```

To move rest close eye images to train folder.

```
!mv /prepared_data/open_eyes/* /prepared_data/train/close_eyes/
```

To move 415 open eye images to test folder.

```
# open eyes
parent_folder = "prepared_data/open_eyes"
files = os.listdir('prepared_data/open_eyes')
for each in range(len(files[:4015])):
  old_path = f"{parent_folder}/{files[each]}"
  new_path = f"prepared_data/test/open_eyes/{files[each]}"
  shutil.move(old_path, new_path)
```

To move rest open eye images to train folder.

```
!mv /prepared_data/open_eyes/* /prepared_data/train/open_eyes/
```

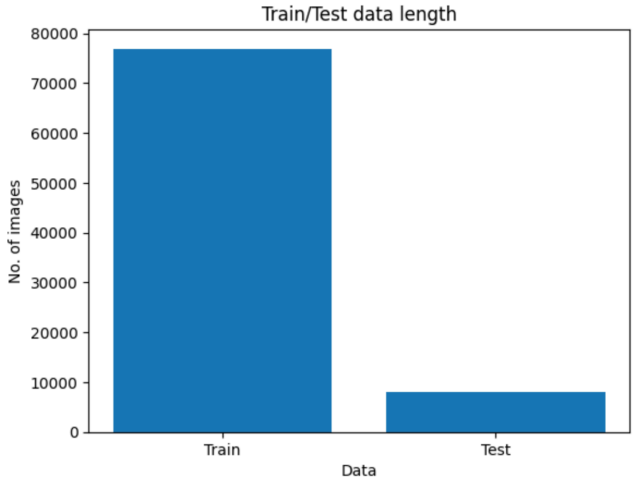

Figure 8. Train / Test Data split

5 Models

5.1 Convolutional Neural Network (CNN)

CNNs are a type of deep learning algorithm that finds wide application in tasks such as image classification, object recognition, and segmentation. They are intended for use with inputs with a grid-like structure, such as photographs or time series data.

Finally, fully connected layers are utilized to perform the ultimate classification of the input image, relying on the features extracted by the convolutional and pooling layers.

The ability of CNNs to understand and adapt features automatically from input data eliminates the requirement for handcrafted feature engineering. Therefore, they are extremely effective for a wide variety of image processing tasks.

Overall, CNNs are a powerful and adaptable deep learning technique that has found widespread application in a wide range of image processing applications. They can obtain cutting-edge performance on a variety of tasks and are anticipated to remain a significant tool in the field of machine learning.

5.2 Xception Model

The Xception architecture [14] is like Inceptionv3, which employs a mix of convolutional layers, pooling layers, and inception modules. Xception, on the other hand, takes this method to its logical conclusion by employing depth wise separable convolutions, a sort of convolutional operation that divides the typical convolutional operation into two independent operations: a depth wise convolution and a pointwise convolution. The number of parameters and computations required is significantly reduced while maintaining or enhancing accuracy on picture classification tasks. Other elements used by Xception to boost performance and

efficiency include batch normalization, residual connections, and a fully linked layer at the network's end.

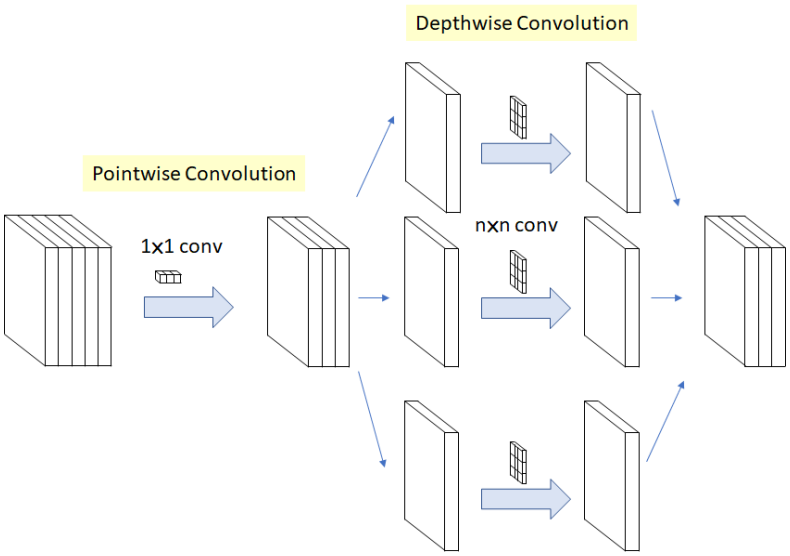

Figure 9. Xception Model Brief Architecture

Overall, Xception is a powerful and efficient CNN architecture that has been shown to be extremely useful for image classification applications. Because of its excellent performance and efficient design, it has been widely employed in both research and industry and has become a popular choice for transfer learning.

We are using transfer learning and considering just the knowledge to be used so we can reduce the size of the model by reducing all the features of the pretrained model.

Below is code snippet for definition of the model.

We are using Xception model as basemodel and further we are Flattening the output of the model and giving activation “relu” and “softmax” to the model before assigning it to the model.

```
xception_base_model = Xception(include_top=False, weights='imagenet', input_tensor=Input(shape=(80,80,3)))
hmodel = xception_base_model.output
hmodel = Flatten()(hmodel)
hmodel = Dense(64, activation='relu')(hmodel)
hmodel = Dropout(0.5)(hmodel)
hmodel = Dense(2, activation= 'softmax')(hmodel)

model = Model(inputs=xception_base_model.input, outputs= hmodel)
for layer in xception_base_model.layers:
    layer.trainable = False
```

The below mentioned code snippet is for defining checkpoints and earlystop in case if the model is stagnant and is not improving after a point. This code also makes sure to save the best model only.

```
checkpoint = ModelCheckpoint(f'{model_folder}xception_model0412.h5',
                             monitor='val_loss',save_best_only=True,verbose=2)

earlystop = EarlyStopping(monitor = 'val_loss', patience=7, verbose= 2, restore_best_weights=True)

learning_rate = ReduceLROnPlateau(monitor = 'val_loss', patience=3, verbose= 2,)

callbacks=[checkpoint,earlystop,learning_rate]
```

The below written code is used for starting the training of the model and storing the history of the progress into the variable history which later can be used to visualize the progress of our training.

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=['accuracy'])

history = model.fit(train_data,steps_per_epoch=train_data.samples//batchsize,
                    validation_data=validation_data,
                    validation_steps=validation_data.samples//batchsize,
                    callbacks=callbacks,
                     epochs=5)
```

5.3 InceptionV3 Model

Inceptionv3 [13] is a convolutional neural network (CNN) architecture created by Google researchers in 2015 as part of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The architecture is intended to be more computationally efficient than earlier Inception network versions while preserving or enhancing image categorization accuracy.

The Inceptionv3 architecture employs a combination of convolutional layers, pooling layers, and inception modules, the latter of which allows the network to learn a variety of unique feature maps at various sizes and resolutions. The inception module combines different-sized filters, allowing the network to capture both fine-grained and coarse-grained features in the same layer.

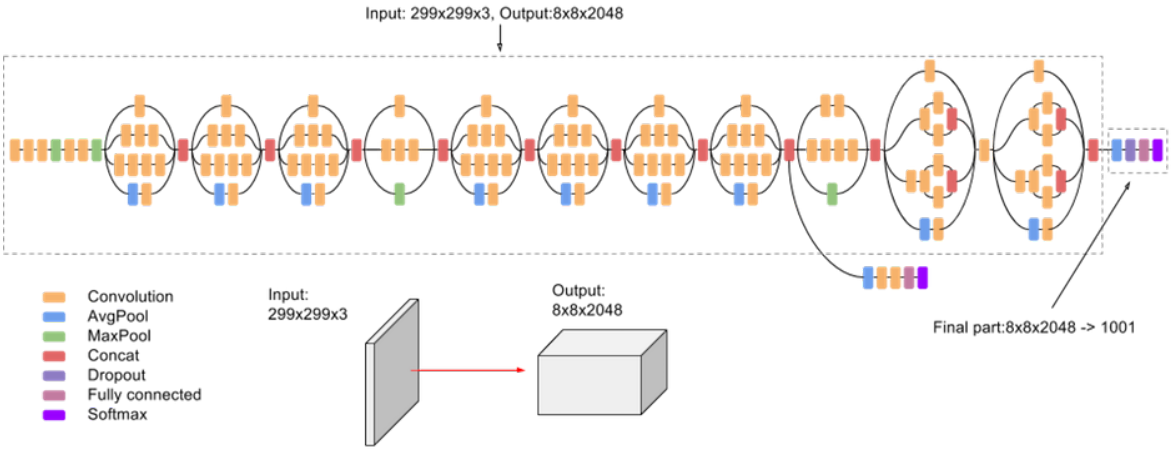

Figure 10. Inception v3

Inceptionv3 is a sophisticated and efficient CNN architecture that has shown to be extremely useful for image classification applications. Because of its excellent performance and efficient design, it has been widely employed in both research and industry and has become a popular choice for transfer learning.

We are using InceptionV3 model as base model and further we are Flattening the output of the model and giving activation “relu” and “softmax” to the model before assigning it to the model.

```
inception_basemodel = InceptionV3(include_top=False, weights='imagenet', input_tensor=Input(shape=(80,80,3)))
hmodel = inception_basemodel.output
hmodel = Flatten()(hmodel)
hmodel = Dense(64, activation='relu')(hmodel)
hmodel = Dropout(0.5)(hmodel)
hmodel = Dense(2,activation= 'softmax')(hmodel)

model = Model(inputs=inception_basemodel.input, outputs= hmodel)
for layer in inception_basemodel.layers:
    layer.trainable = False
```

```
checkpoint = ModelCheckpoint(f'{model_folder}inception3_model0414.h5',
                             monitor='val_loss',save_best_only=True,verbose=2)

earlystop = EarlyStopping(monitor = 'val_loss', patience=7, verbose= 2, restore_best_weights=True)

learning_rate = ReduceLROnPlateau(monitor = 'val_loss', patience=3, verbose= 2,)

callbacks=[checkpoint,earlystop,learning_rate]
```

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=['accuracy'])

history = model.fit(train_data,steps_per_epoch=train_data.samples//batchsize,
                    validation_data=validation_data,
                    validation_steps=validation_data.samples//batchsize,
                    callbacks=callbacks,
                     epochs=10)
```

5.4 Optimizer

In machine learning and deep learning, optimizers are algorithms or methods used to alter the parameters of a model during the training process in order to minimize the error or loss function. An optimizer's main objective is to figure out the best values for a model's parameters that result in the most efficient performance.

Optimizers work by iteratively updating the model parameters based on the gradients of the loss function with respect to the parameters. Gradients represent the steepest increase in the loss function, and the optimizer uses this information to change the parameters so that the loss is minimized. This method is performed several times until the model converges to a point where the loss function is minimized.

- Gradient Descent: A simple optimization approach that changes the parameters in small steps proportionate to the loss function's negative gradient.
- Stochastic Gradient Descent (SGD): A gradient descent variation that updates the parameters at each iteration based on a random selection of training instances, making it faster and more suitable for huge datasets.
- Adam (Adaptive Moment Estimation): An adaptive optimization technique that employs both the first and second moments of the gradients to adaptively alter the learning rate for each parameter, allowing it to handle sparse gradients more efficiently and effectively.
- RMSprop (Root Mean Square Propagation): Another adaptive optimization approach that uses exponential moving averages of squared gradients to adaptively alter the learning rate, allowing it to handle gradients of varying scales.

- Adagrad (Adaptive Gradient Descent): A sparse data-suitable adaptive optimization technique that modifies the learning rate for each parameter based on its previous gradients.

Based on discussions with professor and findings from various articles, I determined that Adam and SGD would be a good combo for the goal of binary class-based image classification, allowing us to detect downiness in real time.

6 Assessment Metrics

6.1 Confusion Matrix

The effectiveness of a classification model can be assessed by comparing predicted class labels and actual class labels using a confusion matrix, which is a tabular representation. In machine learning and data science, it is frequently used to measure the accuracy and quality of a classification model.

The confusion matrix gives information on the accuracy of a model's predictions for classifying data points into different classes.

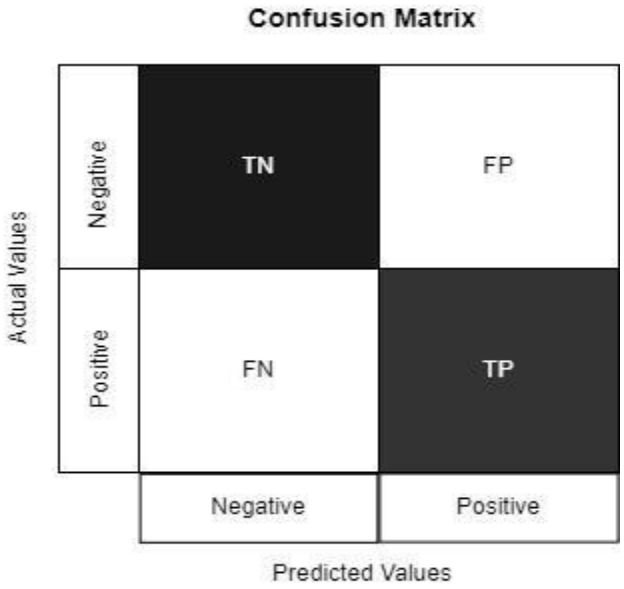

Figure 11. Confusion Matrix

Figure shows the representation of a confusion matrix. One axis of this matrix is the label on one axis reads "Actual values," while the other reads "Predicted values." The confusion matrix

consists of four quadrants, each representing the count of data points based on their predicted and actual class labels. These quadrants provide insights into the performance of the model in terms of accuracy, precision, recall, and other metrics after making predictions on the dataset. The table below explains what they mean.

True Positive (TP)	Instances that are correctly classified as positive by the model.
True Negative (TN)	Instances that are correctly classified as negative by the model.
False Positive (FP)	Instances that are incorrectly classified as positive by the model.
False Negative (FN)	Instances that are incorrectly classified as negative by the model

Table 4 - Performance Metrics for Confusion Matrix

After training model and while trying to determine the accuracy, I calculated Confusion metrics for the test data.

6.2 Precision

The precision score is commonly used in machine learning and statistics to determine the effectiveness of a binary classification model. It is calculated as the ratio of true positive predictions to total positive predictions, showing the proportion of positively predicted cases properly predicted out of all positive instances predicted by the model.

$$\text{Precision} = \frac{TP}{TP + FP}$$

I calculated this Precision score by using the code snippet provided below.

```
from sklearn.metrics import confusion_matrix

# generate sample confusion matrix

conf_mat = [[410, 3],

            [50, 342]]

# calculate precision for class 1 (index 1)

class_idx0 = 0

TP0 = conf_mat[class_idx0][class_idx0]

FP0 = sum(conf_mat[i][class_idx0] for i in range(len(conf_mat))) - TP0

class_idx1 = 1

TP1 = conf_mat[class_idx1][class_idx1]

FP1 = sum(conf_mat[i][class_idx1] for i in range(len(conf_mat))) - TP1

precision0 = TP0 / (TP0 + FP0)

precision1 = TP1 / (TP1 + FP1)

print ("Precision for class ", class_idx0, ": ", precision0)

print("Precision for class ", class_idx1, ": ", precision1)
```

6.3 Recall

In machine learning and statistics, recall score is a regularly used metric for measuring the effectiveness of a model. It is calculated as the ratio of precise positive predictions to total

positive occurrences, or the ratio of correctly predicted positive occurrences among all positive cases in the data.

The recall score formula is as follows:

$$\text{Recall} = \frac{TP}{TP + FN}$$

I used the following code to get recall score for the model we made.

```
from sklearn.metrics import recall_score

recall_score = recall_score(y_test, y_pred)

print("Recall: ", recall_score)
```

6.4 Accuracy

Accuracy is a popular metric in machine learning and statistics for assessing a classification model's performance. It is formulated as the proportion of correct predictions made to total predictions made by the model.

The formula for precision is as follows:

$$\text{Accuracy} = \frac{TN + TP}{TP + TN + FP + FN}$$

Below is the snippet of the code used to obtain accuracy of the model.

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy: ", accuracy)
```

6.5 F1 Score

The F1 score is a machine learning and statistics indicator for evaluating the effectiveness of a classification model. The harmonic mean of precision and recall is the metric that balances these two metrics.

The F1 score formula is as follows:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Below is the snipper of the code used to obtain F1 Score of the model.

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("F1 score: ", f1)
```

6.6 Model Train Accuracy and Loss

The accuracy of a model on the training dataset on which it was trained is referred to as model train accuracy. It quantifies how well the model fits the training data. To calculate model train accuracy, we test the machine learning model's performance on the same dataset that it was trained on. The train accuracy is then calculated as the ratio of properly predicted labels to total labels in the training dataset.

The model train accuracy may be calculated in Python using the Keras model object's evaluate method:

```
loss_tr, acc_tr = model.evaluate(train_data, verbose=2)
print(acc_tr)
print(loss_tr)
```

6.7 Model Test Accuracy

The accuracy of a model on a test dataset is referred to as model test accuracy. It measures how well the model generalizes to new, previously unknown data. To compute the model test accuracy, we first train the machine learning model on a training dataset and then evaluate its performance on a new test dataset. The test accuracy is then calculated as the ratio of properly predicted labels to total labels in the test dataset.

6.8 Model Loss

The model loss in machine learning is a measure of how well the model predicts the proper output for a given input. It is a statistic that is used to determine the performance of a machine learning model during training. A loss function is used to calculate the model loss, which compares the anticipated outcome of the model to the true output and generates a numerical value that represent the difference between them. The purpose of training the model is to minimize the l value of the loss function, indicating that the model is making accurate predictions.

In Python, you can compute the model test accuracy using the evaluate method of the Keras model object:

```
loss_test, accuracy_test = model.evaluate(test_data, verbose=2)
print(accuracy_test)
print(loss_tr)
```

7 Experiments

Model Xception

Training Data: 67276 Validation Data: 16818 Testing Data: 805

Batch size 32

Training Data: 67276 Validation Data: 16818 Testing Data: 805

Train loss: 0.1471 - accuracy: 0.9394

Validation loss: 0.2586 - accuracy: 0.8996

Test loss: 0.3041 - accuracy: 0.8957

```
Epoch 1/5
2102/2102 [==============================] - ETA: 0s - loss: 0.2638 - accuracy: 0.8920
Epoch 1: val_loss improved from inf to 0.30347, saving model to ./models/xception_model0412.h5
2102/2102 [==============================] - 1246s 592ms/step - loss: 0.2638 - accuracy: 0.8920 - val_loss: 0.3035 - val_a
ccuracy: 0.8604 - lr: 0.0010
Epoch 2/5
2102/2102 [==============================] - ETA: 0s - loss: 0.2088 - accuracy: 0.9132
Epoch 2: val_loss improved from 0.30347 to 0.28603, saving model to ./models/xception_model0412.h5
2102/2102 [==============================] - 998s 475ms/step - loss: 0.2088 - accuracy: 0.9132 - val_loss: 0.2860 - val_ac
curacy: 0.8727 - lr: 0.0010
Epoch 3/5
2102/2102 [==============================] - ETA: 0s - loss: 0.2004 - accuracy: 0.9186
Epoch 3: val_loss did not improve from 0.28603
2102/2102 [==============================] - 983s 468ms/step - loss: 0.2004 - accuracy: 0.9186 - val_loss: 0.3065 - val_ac
curacy: 0.8755 - lr: 0.0010
Epoch 4/5
2102/2102 [==============================] - ETA: 0s - loss: 0.1889 - accuracy: 0.9218
Epoch 4: val_loss improved from 0.28603 to 0.28055, saving model to ./models/xception_model0412.h5
2102/2102 [==============================] - 1030s 490ms/step - loss: 0.1889 - accuracy: 0.9218 - val_loss: 0.2805 - val_a
ccuracy: 0.8899 - lr: 0.0010
Epoch 5/5
2102/2102 [==============================] - ETA: 0s - loss: 0.1842 - accuracy: 0.9259
Epoch 5: val_loss improved from 0.28055 to 0.26476, saving model to ./models/xception_model0412.h5
2102/2102 [==============================] - 1025s 487ms/step - loss: 0.1842 - accuracy: 0.9259 - val_loss: 0.2648 - val_a
ccuracy: 0.8959 - lr: 0.0010
```

Figure 12. Xception Model Training Batch size 32 1% Testing Data

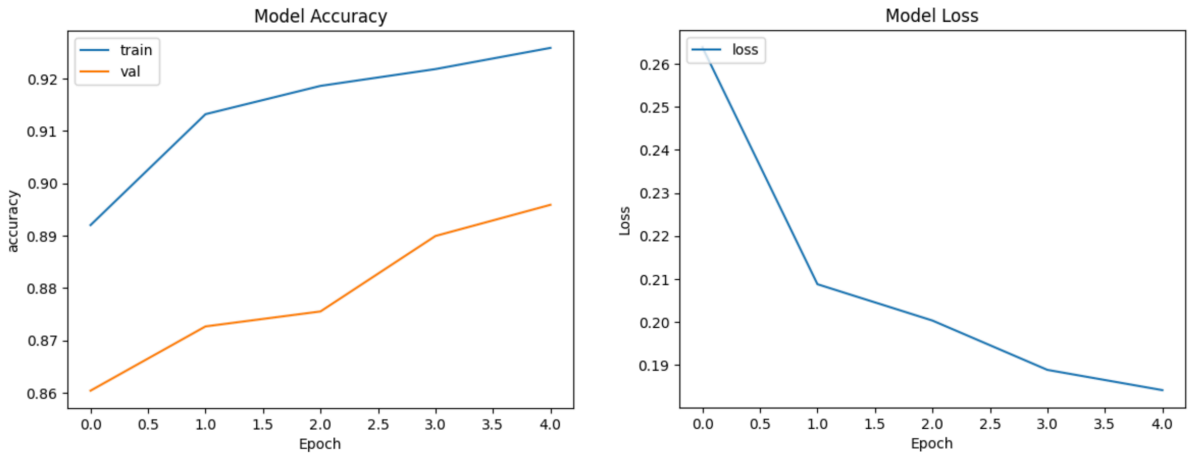

Figure 13. Xception Model Accuracy and Loss Batch Size 32 1% Testing Data

Batch size 8

Training Data: 67276 Validation Data: 16818 Testing Data: 805

Train loss: 0.1479 - accuracy: 0.9402

Validation loss: 0.2592 - accuracy: 0.8974

Test loss: 0.3041 - accuracy: 0.8957

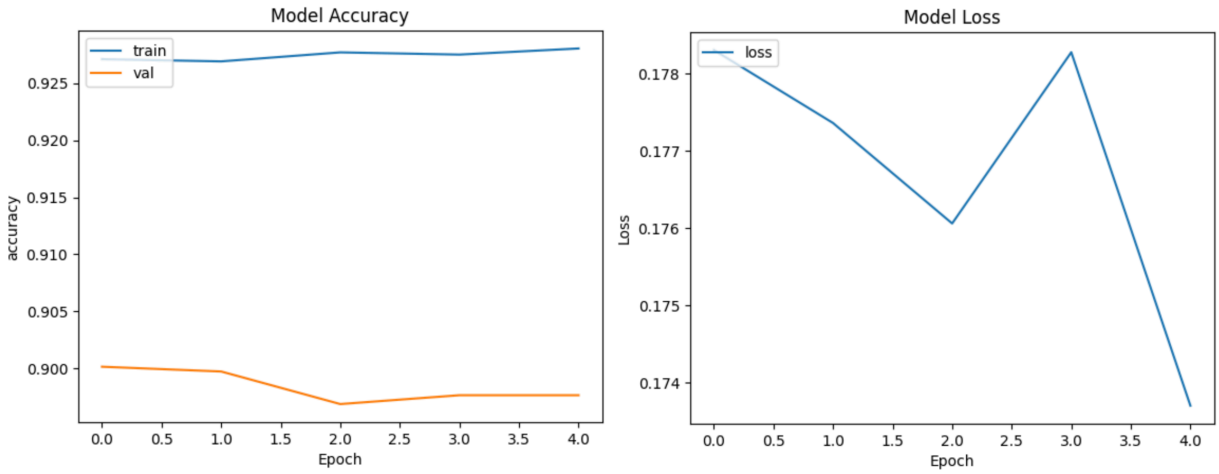

Figure 14. Xception Model Accuracy and Loss 1% Testing Data Tweaked

Batch size 16

Adam Learning Rate 0.001 Default

Training Data: 61516 Validation Data: 15378 Testing Data: 8005

Train loss: 0.1566 - accuracy: 0.9407

Validation Loss: 0.2755 - accuracy: 0.8800

Test loss: 0.1778 - accuracy: 0.9389

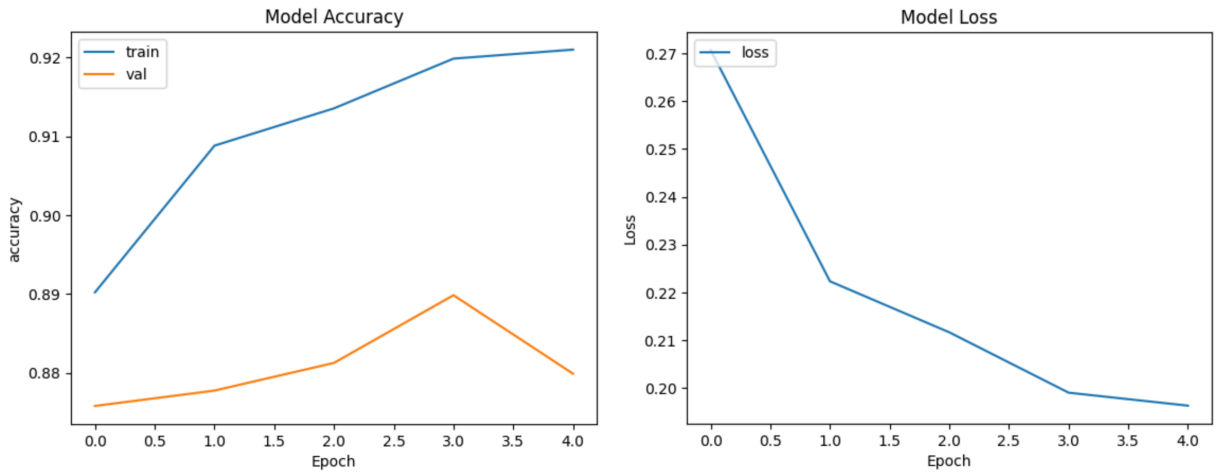

Figure 15. Xception Model Accuracy and Loss Batch Size 16 10% Testing Data

Batch Size 8

Adam learning rate 0.0001

Training Data: 61516 Validation Data: 15378 Testing Data: 8005

Train loss: 0.1591 - accuracy: 0.9401

Validation loss: 0.2728 - accuracy: 0.8832

Test loss: 0.1778 - accuracy: 0.9389

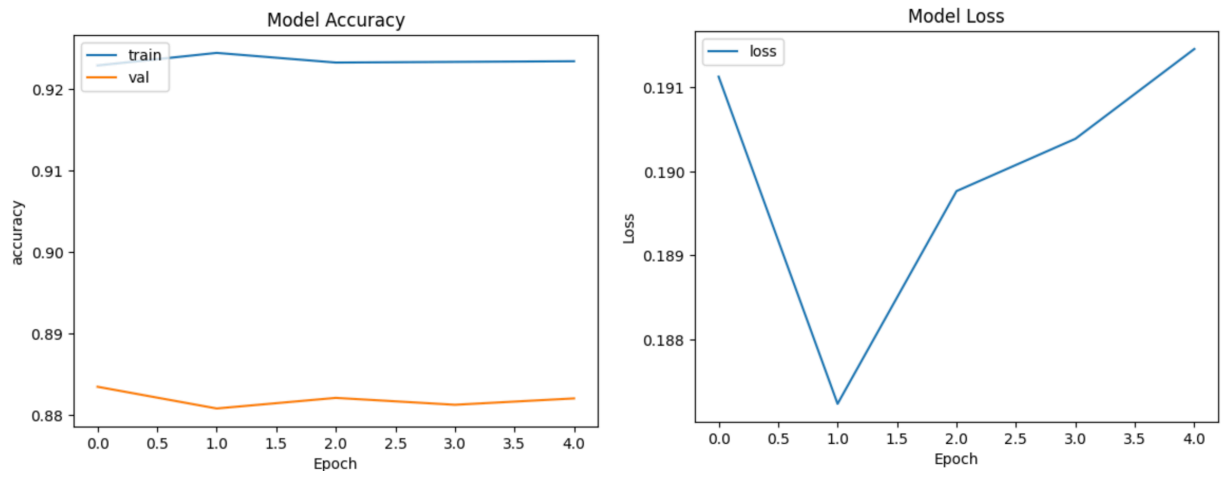

Figure 16. Xception Model Accuracy and Loss 10% Testing Data Tweaked

Model InceptionV3
Batch size 16

Training Data 61516 images

Validation Data 15378 images

Testing Data 8005

Train loss: 0.1506 - accuracy: 0.9406

Validation loss: 0.2452 - accuracy: 0.9011

Test loss: 0.1515 - accuracy: 0.9435

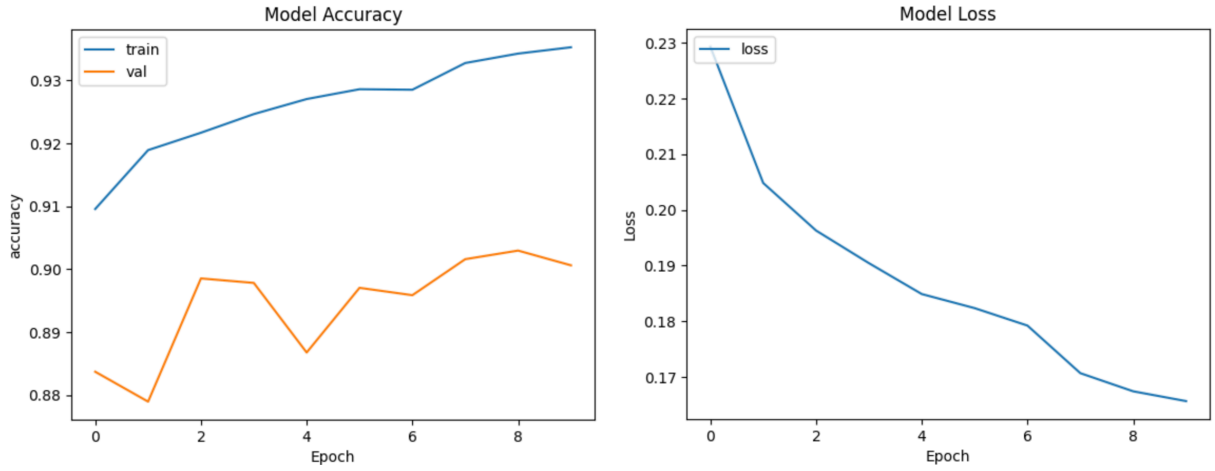

Figure 17. InceptionV3 Model Accuracy and Loss Batch Size 16 10% Testing Data

Batch Size: 8

Training Data 61516 images

Validation Data 15378 images

Test Data 8005 images

Train loss: 0.1217 - accuracy: 0.9685

Validation loss: 0.1883 - accuracy: 0.9354

Test loss: 0.1252 - accuracy: 0.9614

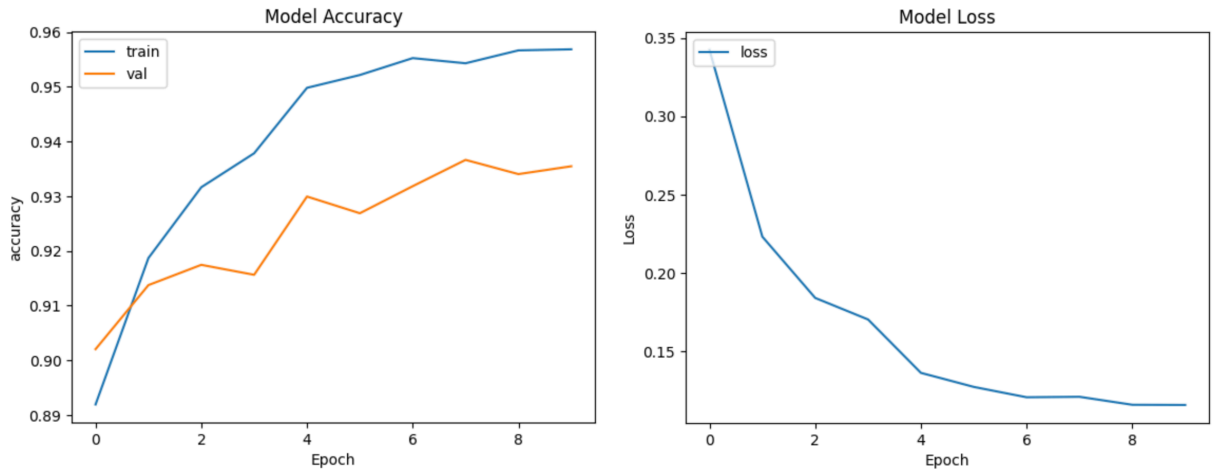

Figure 18. InceptionV3 Model Accuracy and Loss Batch Size 8 10% Testing Data Tweaked

8 Results

We implemented the Inceptionv3 and Xception Model which are based on CNN Architecture.

Results for Xception

My results from the preceding experiments are provided more below:

A big, balanced dataset, as well as the time and effort I put into cleaning and organizing the data, benefited my studies and proved to be one of the most crucial variables in increasing performance ratings.

All the model's results are described in this part, and after reviewing the results, we chose to proceed with the model, and since we are receiving the best results using this model and the specified altering the model using optimization, we decided to move forward with it.

Below is the image representation for the confusion metrics for Xception model.

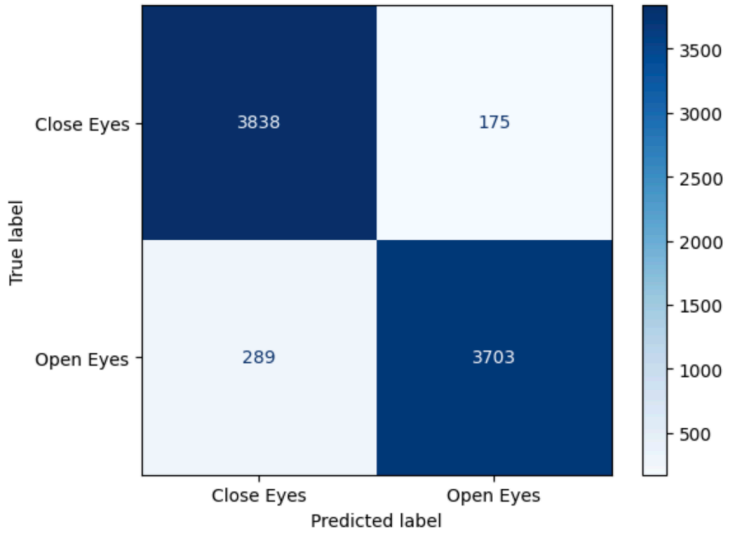

Figure 19. Confusion Metrics for Xception

The precision score is calculated as follows:

Class 0 = Close Eyes

Class 1 = Open Eyes

Precision for class 0: 0.9299733462563605

Precision for class 1: 0.9548736462093863

Recall is calculated using the formula mentioned below.

recall = TP / (TP + FN)

Recall: 0.9276052104208417

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Using the formula for accuracy above we got accuracy of about 0.9420362273579013.

F1 score = 2 * (precision * recall) / (precision + recall)

F1 score: 0.9410419313850064

Results for Inceptionv3

The confusion matrix illustrated in Figure below exhibits a high recall score, indicating that our algorithm consistently made accurate predictions for image classes. Also, the matrix has a high True Negative Rate (TNR), signifying that the algorithm correctly identified both open and close eyes. As a result, the Inceptionv3 model produced good classification results and scored well on the F-1 scale.

Below is the image representation for the confusion metrics for inceptionv3 model.

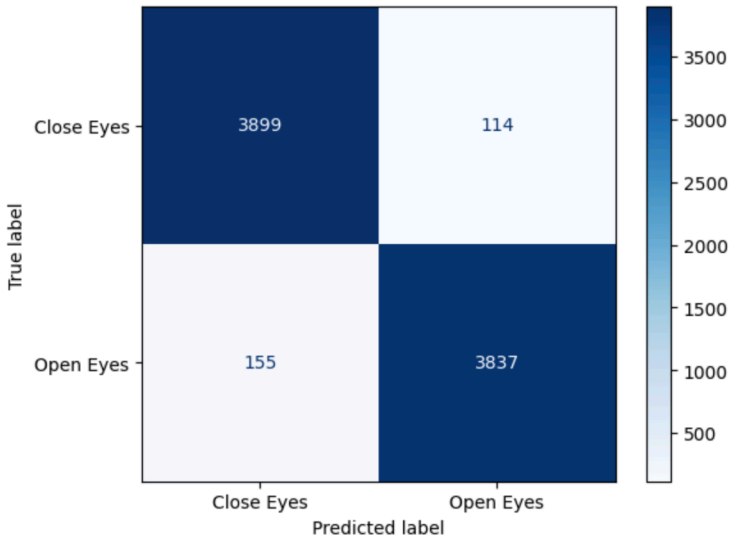

Figure 20. Confusion Metrics for Inceptionv3

The precision score is calculated as follows:

Class 0 = Close Eyes

Class 1 = Open Eyes

Precision for class 0: 0.9617661568820918

Precision for class 1: 0.9711465451784358

Recall is calculated using the formula mentioned below.

recall = TP / (TP + FN)

Recall: 0.9611723446893787

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Using the formula for accuracy above we got accuracy of about 0.9663960024984385.

F1 score = 2 * (precision * recall) / (precision + recall)

F1 score: 0.9661337026312476

9 Conclusion and Future Work

In conclusion, our study investigated the effectiveness of two deep learning models, Xception and Inceptionv3, in drowsiness detection. Through several tests and evaluations, it was determined that Inceptionv3 outperformed Xception in terms of overall performance. The Inceptionv3 model showed better accuracy, precision, recall and F-1 score compared to Xception, indicating its better ability to detect drowsiness from the input data.

Our findings indicate that the architecture and design of the Inceptionv3 model, with its many inception modules and efficient feature extraction capabilities, are well-suited to the task of drowsiness detection. The model's ability to accurately capture and represent the complex patterns associated with drowsiness in the data resulted in improved performance compared to Xception.

While our study shows that Inceptionv3 outperforms Xception for drowsiness detection, there are still several possibilities for further research and development in this field. Further research into various deep learning models, hyperparameter tuning, data augmentation approaches, variability in training data, and real-time implementation may lead to enhanced performance and real-world usability of sleepiness detection systems.

References

1. Arefnezhad, S., Hamet, J., Eichberger, A., Frühwirth, M., Ischebeck, A., Koglbauer, I. V., Moser, M., & Yousefi, A. (2022, February 16). *Driver drowsiness estimation using EEG signals with a dynamical encoder–decoder modeling framework*. Nature News. Retrieved April 5, 2023, from https://www.nature.com/articles/s41598-022-05810-x
2. Lee, S., & Lee, J. (2018). Non-intrusive driver fatigue monitoring system using yawning extraction. In 2018 2nd International Conference on BioSignal Analysis, Processing and Systems (ICBAPS) (pp. 1-5). IEEE. https://www.hindawi.com/journals/ijvt/2014/678786/
3. Arefnezhad, S., Hamet, J., Eichberger, A., Frühwirth, M., Ischebeck, A., Koglbauer, I. V., Moser, M., & Yousefi, A. (2022, February 16). *Driver drowsiness estimation using EEG signals with a dynamical encoder–decoder modeling framework*. Nature News. Retrieved April 5, 2023, from https://www.nature.com/articles/s41598-022-05810-x
4. *Driver drowsiness detection using EEG power spectrum analysis*. (n.d.). Retrieved April 6, 2023, from https://ieeexplore.ieee.org/abstract/document/6863035/
5. Mehta, S., Dadhich, S., Gumber, S., & Jadhav Bhatt, A. (2019, June 14). *Real-time driver drowsiness detection system using eye aspect ratio and eye closure ratio*. SSRN. Retrieved April 5, 2023, from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3356401
6. Awais, M., Badruddin, N., & Drieberg, M. (n.d.). *Driver drowsiness detection using EEG power spectrum analysis*. Retrieved April 6, 2023, from https://ieeexplore.ieee.org/abstract/document/6863035/

7. Ngxande, M., Burke, M., & Tapamo, J.-R. (n.d.). *Driver drowsiness detection using behavioral measures and machine ...* Retrieved April 6, 2023, from https://ieeexplore.ieee.org/document/8261140
8. Shakeel, M. F., Bajwa, N. A., Anwaar, A. M., Sohail, A., Khan, A., & Haroon-ur-Rashid. (1970, January 1). *Detecting driver drowsiness in real time through deep learning based object detection*. SpringerLink. Retrieved April 6, 2023, from https://link.springer.com/chapter/10.1007/978-3-030-20521-8_24
9. Rafid, A.-U.-I., Chowdhury, A. I., Niloy, A. R., & Sharmin, N. (n.d.). *A deep learning based approach for real-time driver ... - IEEE xplore*. Retrieved April 6, 2023, from https://ieeexplore.ieee.org/abstract/document/9667944/
10. DataTurks. (2018, July 12). *Face detection in images*. Kaggle. Retrieved April 6, 2023, from https://www.kaggle.com/datasets/dataturks/face-detection-in-images
11. Gupta, A. (2020, September 21). *Human faces*. Kaggle. Retrieved April 6, 2023, from https://www.kaggle.com/datasets/ashwingupta3012/human-faces
12. *MRL Eye Dataset*. MRL. (n.d.). Retrieved April 6, 2023, from http://mrl.cs.vsb.cz/eyedataset
13. Google. (n.d.). *Advanced guide to inception V3 | cloud TPU | google cloud*. Google. Retrieved April 7, 2023, from https://cloud.google.com/tpu/docs/inception-v3-advanced
14. Tsang, S.-H. (2019, March 20). *Review: Xception - with depthwise separable convolution, better than inception-V3 (image...* Medium. Retrieved April 7, 2023, from https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568