ABSTRACT

On an average 1200 road accidents record daily in India out of which 400 leads to direct death and rest gets effected badly. The major reason of these accidents is drowsiness caused by both sleep and alcohol. Due to driving for long time or intoxication, drivers might feel sleepy which is the biggest distraction for them while driving. This distraction might cost death of driver and other passengers in the vehicle and at the same time it also causes death of people in the other vehicles and pedestrians too. This mistake of one person on road would take their own life and also takes lives of other and put respective families in sorrow and tough situations.

To prevent such accidents we, team 5A propose a system which alerts the driver if he/she feels drowsy. To accomplish this, we implement the solution using computer-vision based machine learning model. The driver's face is detected by face recognition algorithm continuously using a camera and the face of the driver is captured. The face of the driver is given as input to a classification algorithm which is trained with a data set of images of drowsy and non-drowsy faces. The algorithm uses landmark detection to classify the face as drowsy or not drowsy. If the driver's face is drowsy, a voice alert is generated by the system. This alert can make the driver aware that he/she is feeling drowsy and the necessary actions can then be taken by the driver. This system can be used in any vehicle on the road to ensure safety of the people who are travelling and prevent accidents which are caused due to the drowsiness of the driver.

Keywords: Computer Vision, Machine Learning, Convolutional Neural Networks.

1. INTRODUCTION

1.1. Introduction

Car accident is the major cause of death in which around 1.3 million people die every year. Majority of these accidents are caused because of distraction or the drowsiness of driver. The countless number of people drives for long distance every day and night on the highway. Drowsiness appears in situations of stress and fatigue in an unexpected and inopportune way, and it may be produced by sleep disorders, certain type of medications, and even, boredom situations, for example, driving for a long time. In this way, drowsiness produces danger situations and increases the probability that an accident occurs. In this context, it is important to use new technologies to design and to build systems that will monitor drivers, and measure their level of attention throughout the whole driving process. To prevent such accidents, our team has come up with a solution for this. In this system, a camera is used to record user's visual characteristics. We use face detection and CNN techniques and try to detect the drowsiness of driver, if he/she is drowsy then alarm will be generated. So that the driver will get cautious and take preventive measures. Driver drowsiness detection contributes to the decrease in the number of deaths occurring in traffic accident.

1.2. Objective

Traffic accidents due to human errors cause many deaths and injuries around the world. The major cause of these accidents is drowsiness of the driver due to sleeplessness or long driving hours. There is need for a system developed with the technologies that are available today which can overcome this situation. The aim of this system is to reduce the number of accidents by developing a model which can generate an alert if the driver is feeling drowsy so that the driver can become aware and take necessary actions.

1.3 Future Scope:

The work can be extended by combing physiological approach techniques, so the driver can be detected as drowsy through those readings. In special situations like when a driver wares spectacles along with mask then we can use the physiological reading of the driver. We can detect the drowsiness of the driver more accurately with these readings. We can use the devices which are mentioned earlier and detect the drowsiness of the driver. While this is a research project, there is scope when this completely turns out to be developed into an application which can be run by the end users on their own for their own purposes on their own systems.

1.4. Requirements

1.4.1. Software Requirements:

These are the Software Configurations that are required.

- Operating System: Linux, Windows, Mac OS
- Language: Python 3
- IDE: Jupiter Notebook

1.4.2. Hardware Requirements:

These are the Hardware Configurations that are required.

- Processor
- Ram: 4GB
- Webcam
- Speaker

Chapter 2 - System Analysis & Requirements Specifications

2.1 -System Analysis:

2.1.1 - Paper 1

Title: Drowsiness Detection System Utilizing Physiological Signals.
Author: Trupti K. Dange, T. S. Yengatiwar.
Year of publication: 2013.
Keywords: EOG, ECG, EEG, HRV, SVM, driver drowsiness detection.

The Physiological parameters-based techniques detect drowsiness based on drivers' physical conditions such as heart rate, pulse rate, breathing rate, respiratory rate and body temperature, etc. These biological parameters are more reliable and accurate in drowsiness detection as they are concerned with what is happening with driver physically. Fatigue or drowsiness, change the physiological parameters such as a decrease in blood pressure, heart rate and body temperature, etc. Physiological parameters-based drowsiness detection systems detect these changes and alert the driver when he is in the state, near to sleep. A list of physiological condition-based drowsiness detection system. These measures are invasive, so require electrodes to be directly placed on the driver's body.

1) EEG-BASED DRIVER FATIGUE DETECTION

The drivers' fatigue detection system using Electroencephalogram (EEG) signals is proposed to avoid the road accidents usually caused due to drivers' fatigue. The proposed method firstly finds the index related to different drowsiness levels. The system takes EEG signal as input which is calculated by a cheap single electrode neuro signal acquisition device. To evaluate the proposed method, data set for simulated car driver under the different levels of drowsiness is collected locally. And result shows that the proposed system can detect all subjects of tiredness.

2) WAVELET ANALYSIS OF HEART RATE VARIABILITY & SVM CLASSIFIER

Li and Chung [21] proposed the driver drowsiness detection that uses wavelet analysis of Heart Rate Variability (HRV) and Support Vector Machine (SVM) classifier. The basic purpose is to categorize the alert and drowsy drivers using the wavelet transform of HRV signals over short durations. The system firstly takes Photo Plethysmo Graphy (PPG) signal as input and divide it into 1-minute intervals and then verify two driving events using average percentage of eyelid closure over pupil over time (PERCLOS) measurement over the interval. Secondly, the system performs the feature extraction of HRV time series based on Fast Fourier Transform (FFT) and wavelet. A Receiver Operation Curve (ROC) and SVM classifier is used for feature extraction and

classification respectively. The analysis of ROC shows that the wavelet-based method gives improved results than the FFT-based method. Finally, the real time requirements for drowsiness detection, FFT and wavelet features are used to train the SVM classifier extracted from the HRV signals.

3) PULSE SENSOR METHOD

Mostly, previous studies focus on the physical conditions of drivers to detect drowsiness. That's why Rahim detects the drowsy drivers using infrared heart-rate sensors or pulse sensors. The pulse sensor measures the heart pulse rate from drivers' finger or hand. The sensor is attached with the finger or hand, detects the amount of blood flowing through the finger. Then amount of the blood's oxygen is shown in the finger, which causes the infrared light to reflect off and to the transmitter. The sensor picks up the fluctuation of oxygen that are connected to the Arduino as microcontroller. Then, the heart pulse rate is visualizing by the software processing of HRV frequency domain. Experimental results show that LF/HF (Low to high frequency) ratio decreases as drivers go from the state of being awake to the drowsy and many road accidents can be avoided if an alert is sent on time.

4) WEARABLE DRIVER DROWSINESS DETECTION SYSTEM

Mobile based applications have been developed to detect the drowsiness of drivers. But mobile phones distract the drivers' attention and may cause accident. To address the issue, Lenget proposed the wearable-type drowsiness detection system. The system uses selfdesigned wrist band consists of PPG signal and galvanic skin response sensor. The data collected from the sensors are delivered to the mobile device which acts as the main evaluating unit. The collected data are examined with the motion sensors that are built-in in the mobiles. Then five features are extracted from the data: heart rate, respiratory rate, stress level, pulse rate variability, and adjustment counter. The features are moreover used as the computation parameters to the SVM classifier to determine the drowsiness state. The experimental results show that the accuracy of the proposed system reaches up to 98.02 %. Mobile phone generates graphical and vibrational alarm to alert the driver.

5) WIRELESS WEARABLES METHOD

To avoid the disastrous road accidents, Warwick proposed the idea for drowsiness detection system using wearable Bio sensor called Bio-harness. The system has two phases. In the first phase, the physiological data of driver is collected using bio-harness and then analyzes the data to find the key parameters like ECG, heart rate, posture and others related to the drowsiness. In the second phase, drowsiness detection algorithm will be designed and develop a mobile app to alert the drowsy drivers.

6) DRIVER FATIGUE DETECTION SYSTEM

Chellappa presents the Driver fatigue detection system. The basic of the system is to detect the drowsiness when the vehicle is in the motion. The system has three components: external hardware (sensors and camera), data processing module and alert.

unit. Hardware unit communicates over the USB port with the rest of the system. Physiological and physical factors like pulse rate, yawning, closed eyes, blink duration and others are continuously monitored using somatic sensor. The processing module uses the combination of the factors to detect drowsiness. In the end, alert unit alerts the driver at multiple stages according to the severity of the symptoms.

7) HYBRID APPROACH UTILIZING PHYSIOLOGICAL FEATURES

To improve the performance of detection, Awais proposed the hybrid method which integrates the features of ECG and EEG. The method firstly extracts the time and frequency domain features like time domain statistical descriptors, complexity measures and power spectral measures from EEG. Then using ECG, features like heart rate, HRV, low frequency, high frequency and LF/HF ratio. After that, subjective sleepiness is measured to study its relationship with drowsiness. To select only statistically significant features, t-tests is used that can differentiate between the drowsy and alert. The features extracted from ECG and EEG are integrated to study the improvements in the performance using SVM. The other main contribution is to study the channel reduction and its impact on the performance of detection. The method measures the differences between the drowsy and alert state from physiological data collected from the driving simulated-based study. Monotonous driving environment is used to induce the drowsiness in the participants. The proposed method demonstrated that combining ECG and EEG improves the performance of system in differentiating the drowsy and alert states, instead of using them alone. The analysis of channel reduction confirms that the accuracy level reaches to 80% by just combining the ECG and EEG. The performance of the system indicates that the proposed system is feasible for practical drowsiness detection system.

2.1.2 - Paper 2

Title: Drowsiness Detection with OpenCV (Using Eye Aspect Ratio)
Author: Adrian Rosebrock.
Year of publication: 2017.
Keywords: EAR, SVM, eye blink detection.

A real-time algorithm to detect eye blinks in a video sequence from a standard camera is proposed. Recent landmark detectors, trained on in-the wild datasets exhibit excellent robustness against a head orientation with respect to a camera, varying illumination and facial expressions. We show that the landmarks are detected precisely enough to reliably estimate the level of the eye opening. The proposed algorithm therefore estimates the landmark positions, extracts a single scalar quantity – eye aspect ratio (EAR) – characterizing the eye opening in each frame.

6 Many methods have been proposed to automatically detect eye blinks in a video sequence. Several methods are based on motion estimation in the eye region. Typically, the face and eyes are detected by a Viola-Jones type detector. Next, motion in the eye area is estimated from optical flow, by sparse tracking, or by frame-to-frame intensity differencing and adaptive thresholding. Finally, a decision is made whether the eyes are or are not covered by eyelids.

Nowadays, robust real-time facial landmark detectors that capture most of the characteristic points on a human face image, including eye corners and eyelids, are available, Most of the state-of-the-art landmark detectors formulate a regression problem, where a mapping from an image into landmark positions or into other landmark parametrization is learned. These modern landmark detectors are trained on "in-the-wild datasets" and they are thus robust to varying illumination, various facial expressions, and moderate non-frontal head rotations.

Proposed method The eye blink is a fast closing and reopening of a human eye. Each individual has a little bit different pattern of blinks. The pattern differs in the speed of closing and opening, a degree of squeezing the eye and in a blink duration. The eye blink lasts approximately 100-400 ms. We propose to exploit state-of-the-art facial landmark detectors to localize the eyes and eyelid contours. From the landmarks detected in the image, we derive the eye aspect ratio (EAR) that is used as an estimate of the eye opening state. Since the per frame EAR may not necessarily recognize the eye blinks correctly, a classifier that takes a larger temporal window of a frame into

account is trained. The EAR is mostly constant when an eye is open and is getting close to zero while closing an eye.

A real-time eye blink detection algorithm was presented. We quantitatively demonstrated that regression-based facial landmark detectors are precise enough to reliably estimate a level of eye openness. While they are robust to low image quality (low image resolution in a large extent) and in-the-wild phenomena as non-frontality, bad illumination, facial expressions, etc. The proposed SVM method that uses a temporal window of the eye aspect ratio (EAR), outperforms the EAR thresholding.

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

2.1.3-Paper 3

Title: Real Time Driver Fatigue Detection Based on SVM Algorithm.
Authors: Burcu Kir Savas, Yasar Becerkli.
Year of publication: 2018.
Keywords: fatigue detection , SVM , driving safety , video processing PROPOSED SYSTEM

In this study, SVM based driver fatigue prediction system is proposed to increase driver safety. The proposed system has five stages: PERCLOS, count of yawn, internal zone of the mouth opening, count of eye blinking and head detection to extract attributes from video recordings. The classification stage is done with Support Vector Machine (SVM). While the YawDD dataset is used during the training phase of the classification, real-time video recordings are used during the test phase.

SVM is a classification algorithm separating data items. This algorithm proposed by Vladimir N. Vapnik based on statistical learning theory. SVM, one of the machine learning methods, is widely used in the field of pattern recognition. The main purpose of the SVM is to find the best hyperplane to distinguish the data given as two-class or multiclass. The study was carried out in two classes. Whereas label 0 means that the driver is tired, label 1 means the driver is non-tired. Thus, it is aimed to distinguish tired drivers (driver fatigue) from non-tired drivers.

In training driving simulation experiments, we had 10 volunteers (5 men and 5 women) whose ages were between 18-30. All the volunteers drove training simulation during the experiments. Attributions obtained in real time during the driving when fatigue detection system was working are indicated. The results of driver fatigue detection were classified using SVM classification algorithm. The total 713 datas line each of which has five stage features were sampled from 10 volunteers. All the data in the study are divided into two groups: 80% training data set (568 data set) and 20% test data set (145 data set). In the study, cross validation was selected as 10. Measurements are as follows:

• TP means that a real fatigue is detected;

• TN means that a non-fatigue situation is correctly detected as non-fatigue by the system;

• FP means that a non-fatigue situation is incorrectly detected as real fatigue;

• FN means that a real fatigue situation is incorrectly detected as non-fatigue.

The accuracy of driver fatigue calculated as Accuracy= (TP+TN)/(TP+TN+FP+FN)

As a result of the study, a system was designed to investigate the effects of fatigue and insomnia on drivers physical transient behaviors, and to simulate drowsy drivers as a result of research. In this system, behavioral detection model is used for detecting fatigue drivers. The proposed system has five stages: PERCLOS, count of yawn, internal zone of the mouth opening, count of eye blinking and head detection to extract attributes from video recordings. As a result, the system is classified as SVM in two classes (not fatigue / fatigue). In this study a Real Time Driver Fatigue Detection SVM Based on SVM Algorithm is presented whose test] results showed that the accuracy rate of fatigue detection is up to 97.93%

2.1.4- Paper 4

Title: Driver drowsiness detection using ANN image processing.
Authors: T. Vesselenyi1 , S. Moca1 , A. Rus1 , T. Mitran1 , B. Tătaru1.
Keywords: EEG, EOG, ANN, Deep Learning
Year of publication: 2017.

The study regarding the possibility to develop a drowsiness detection system for car drivers based on three types of methods: EEG and EOG signal processing and driver image analysis

EEG (Electroencephalography) and EOG (Electrooculography) signals measurement and on the eye state (closed or opened) image classification. The EEG method monitors the brain activity through a sensor placed on a specific part of the scalp, The EOG method tracks the eye movements by measuring the signals from the muscles which are acting on the eye The eye image analysis can monitor the opened or the closed state of the eye .

The EEG and EOG sensors, electrodes which have to be fixed with a conductive gel and in most devices must transmit the signal by wire, present a major discomfort. Research in the field of advanced materials and MEMS technology may solve these problems, as for example the use of dry electrodes for EEG.

EEG Developments field are supported by efforts to create brain – computer interfaces for different applications, including devices that help disabled people. In this research the central point is to distinguish between low and high alpha rhythm peaks, which can make the difference between alert and drowsy states.

To use EOG signals acquisitioned from 3 sensors (EOG1, EOG2, EOG3). After preprocessing, four types of different signals were identified. The combination of these four types of signals give the information to distinguish between left, right, up and down movements of the eye.

Drowsiness detection using the processing of the driver’s eye images

For the classification of the driver’s drowsy or alert state, artificial neural networks were used. Artificial neural networks are extensively used for the image classification. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Deep Belief Networks, Restricted Boltzmann Machines and Deep Autoencoders are all methods belonging to the Deep Learning. These methods are used in a wide range

of applications, the image classification being one of the fields in which these are employed with success.

Matlab Neural Network Toolbox with 1 layer ANN and the autoencoder module of the Deep Learning Toolbox were used in order to study if these methods can be applied for image classification of the driver's drowsiness. In order to analyze the drowsiness state of the driver 200 images of a driver during a regular driving process were acquisitioned. One hundred of these images contain opened eyes or half opened eyes images and another hundreds of the images contain closed eyes images. It was assumed that the drowsiness is linked to the images in which the driver has closed eyes, and the alert state is linked to the images in which the driver has opened eyes.

2.1.5-Paper 5

Title: Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State. Authors: Venkata Rami Reddy Chirra, Srinivasulu Reddy Uyyala and Venkata Krishna Kishore Kolli. Year of publication: 2019. Keywords: Deep CNN, Feature extraction

PROPOSED DEEP CNN BASED DROWSINESS DETECTION SYSTEM

Proposed system algorithm

(1) Viola-jones face detection algorithm is used to detect the face the images and given as input to Viola-jones eye detection algorithm

(2) Once the face is detected, Viola-jones eye detection algorithm is used to extract the eye region from the facial images and given as input to CNN.

(3) CNN with four convolutional layers is used to extract the deep features and those features are passed to fully connected layer.

(4) Soft max layer in CNN classify the images in to sleepy or non-sleepy images.

Face detection and eye region extraction

Whole face region may not be required to detect the drowsiness but only eyes region is enough for detecting drowsiness. At first step by using the Viola-jones face detection algorithm face is detected from the images. Once the face is detected, Viola-jones eye detection algorithm is used to extract the eye region from the facial images. it is the first algorithm used for face detection. For the face detection the Viola-Jones algorithm having three techniques those are Haar-like features, Ada boost and Cascade classifier. In this work, Viola-Jones object detection algorithm with Haar cascade classifier was used and implemented using OPEN CV with python. Haar cascade classifier uses Haar features for detecting the face from images. Feature extraction and classification.

Feature extraction is one type of dimensionality reduction where useful parts of an image represented as a feature vector.

Convolutional neural network

Convolutional neural network (CNN) is used in the proposed system for detection of driver drowsiness. Since a feature vector is needed for each drowsy image to compare with existing features in a database to detect either drowsy or not. Usually CNNs requires fixed size images as input so preprocessing is required. The preprocessing includes extracting the key frames from video based on temporal changes and store in database. From these stored images, feature vectors are generated in convolution layers of CNN. These feature vectors are then used for the detecting the driver

drowsiness. CNN have layers like convolutional layers, pooling (max, min and average) layers, ReLU layer and fully connected layer. Convolution layer is having kernels (filters) and each kernel having width, depth and height. This layer produces the feature maps as a result of calculating the scalar product between the kernels and local regions of image. CNN uses pooling layers.

(Max or Average) to minimize the size of the feature maps to speed up calculations. In this layer, input image is divided into different regions then operations are performed on each region. In Max Pooling, a maximum value is selected for each region and places it in the corresponding place in the output. ReLU (Rectified Linear Units) is a nonlinear layer. The ReLU layer applies the max function on all the values in the input data and changes all the negative values to zero. The following equation shows the ReLU activation function.

Motivation

In this proposed work a new method is proposed for driver drowsiness detection based on eye state. This determines the state of the eye that is drowsy or non- drowsy and alert with an alarm when state of the eye is drowsy. Face and eye region are detected using ViolaJones detection algorithm. Stacked deep convolution neural network is developed to extract features and used for learning phase. A SoftMax layer in CNN classifier is used to classify the driver as sleep or non-sleep. Proposed system achieved 96.42% accuracy. Proposed system 465 effectively identifies the state of driver and alert with an alarm when the model predicts drowsy output state continuously. In future we will use transfer learning to improve the performance of the system

2.2 -Requirements Specifications:

2.2.1 - OpenCV

(Open Source Computer Vision Library) OpenCV is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

2.2.2 - Numpy

NumPy stands for Numerical Python. NumPy was created in 2005 by Travis Oliphant. NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. At the core of the NumPy package, is the ndarray object. It provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures. NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

2.2.3 - Tensorflow

TensorFlow is an open-source software library. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. Google open-sourced TensorFlow in November 2015.

TensorFlow's popularity is due to many things, but primarily because of the computational graph concept, automatic differentiation, and the adaptability of the Tensorflow python API structure. This makes solving real problems with TensorFlow accessible to most programmers. Google's Tensorflow engine has a unique way of solving problems. This unique way allows for solving machine learning problems very efficiently.

What is Tensor in Tensorflow

TensorFlow, as the name indicates, is a framework to define and run computations involving tensors. A tensor is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes. Each element in the Tensor has the same data type, and the data type is always known. The shape (that is, the number of dimensions it has and the size of each dimension) might be only partially known. Most operations produce tensors of fullyknown shapes if the shapes of their inputs are also fully known, but in some cases it's only possible to find the shape of a tensor at graph execution time.

2.2.4 - Keras

Keras is a python based open-source library used in deep learning (for neural networks). Keras was released in March 2015. It can run on top of TensorFlow, Microsoft CNTK or Theano. It is very simple to understand and use, and suitable for fast experimentation. It is designed to be fast and easy for the user to use. Keras models can be run both on CPU as well as GPU. Keras is the best platform out there to work on neural network models. The API that Keras has a user-friendly where a beginner can easily understand. Keras has the advantage that it can choose any libraries which support it for its backend support. Keras provides various pre-trained models which help the user in further improving the models the user is designing.

2.2.5 - Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc

2.2.6 - Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its datasetoriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Seaborn Heatmap

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

2.2.7 - Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. It was originally called scikits.learn and was initially developed by David Cournapeau as a Google summer of code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation), took this project at another level and made the first public release (v0.1 beta) on 1st Feb. 2010.

Sklearn Metrics

Sklearn provides metrics for evaluating the performance of the model.

Classification Metrics The sklearn.metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values. Most implementations allow each sample to provide a weighted contribution to the overall score, through the sample_weight parameter.

The **confusion_matrix** function evaluates classification accuracy by computing the confusion matrix with each row corresponding to the true class (Wikipedia and other references may use different convention for axes). By definition, entry i,j in a

confusion matrix is the number of observations actually in group i, but predicted to be in group j. Here is an example: Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples.

2.2.8 - OS module in Python

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module. OS comes under Python's standard utility modules. The *os* and *os.path* modules include many functions to interact with the file system.

os.listdir()

os.listdir() method in Python is used to get the list of all files and directories in the specified directory. If we don't specify any directory, then list of files and directories in the current working directory will be returned.

os.path module

This module contains some useful functions on pathnames. The path parameters are either strings or bytes. These functions here are used for different purposes such as for merging, normalizing and retrieving path names in python. All of these functions accept either only bytes or only string objects as their parameters. The result is an object of the same type, if a path or file name is returned. As there are different versions of operating system so there are several versions of this module in the standard library. os.path.join() method in Python join one or more path components intelligently. This method concatenates various path components with exactly one directory separator ('/') following each non-empty part except the last path component. If the last path component to be joined is empty then a directory separator ('/') is put at the end. If a path component represents an absolute path, then all previous components joined are discarded and joining continues from the absolute path component.

2.2.9 - Playsound module in Python

The playsound module is a cross platform module that can play audio files. This doesn't have any dependencies, simply install with pip in your virtual environment and run. Implementation is different on platforms. It uses windll.winm on Windows, AppKit.NSSound on Apple OS X and GStreamer on Linux. The playsound module contains a function named playsound(). It works with both WAV and MP3 files.

2.2.10. SYSTEM ARCHITECTURE

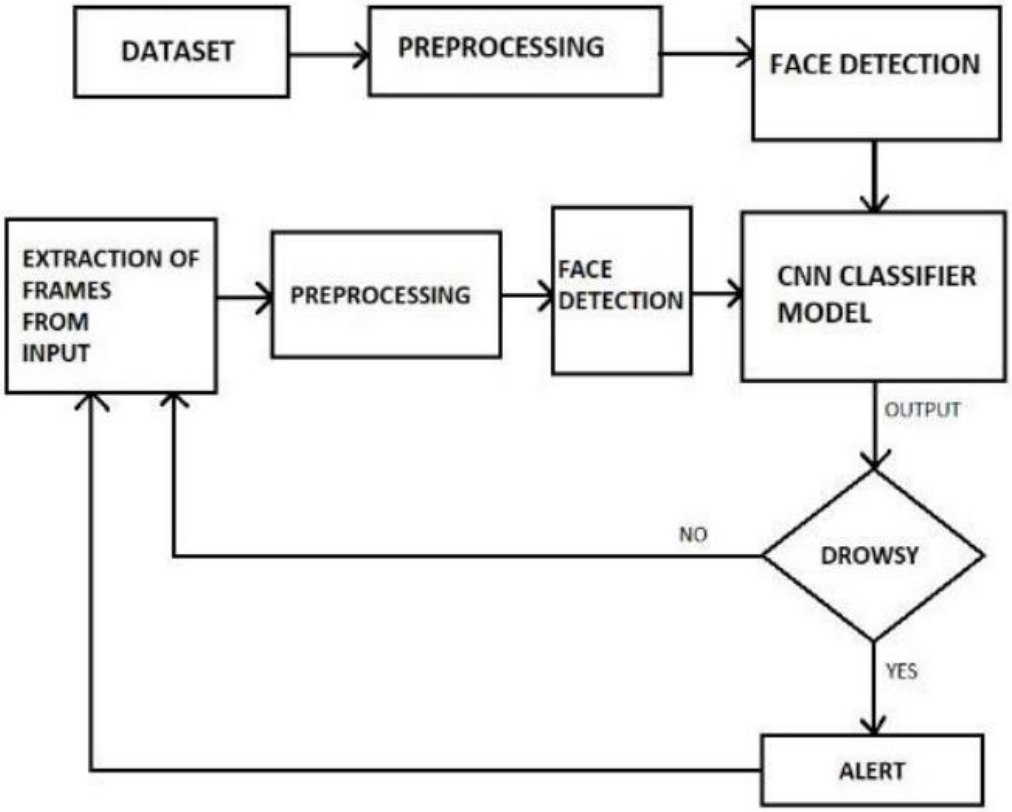

Fig. 2.2.1 System Architecture

MODULES

2.2.11 - Collecting the Dataset

The first step is collecting the dataset. To work with machine learning projects, we need a huge amount of data, because, without the data, one cannot train ML/AI models. Collecting and preparing the dataset is one of the most crucial parts while creating an ML/AI project.

A dataset called Drowsiness_dataset from Kaggle[] which contains images with notyawning faces and yawning faces. This dataset contains images of many people captured while driving when they are yawning and not-yawning. This dataset also includes images with the people wearing spectacles. Another dataset from Github is considered in which the images of 3 people are recorded while driving. This dataset contains 3 sets of images which are alert images, images with closed eyes and images with yawning faces. The images of closed eyes and yawning faces are combined as drowsy images. We combined these images into 2 sets of images i.e., alert and drowsy.

2.2.12 - Face Detection Module

In this module, all the images in the dataset are preprocessed. We have the images for 2 classes (alert and drowsy) in two directories. We use the “os” module to list the directory names and list all the image files in each directory. Then, we can access all the image files in each directory. We use OpenCV to read and resize the images. Instead of giving the entire image to classification model, only the face region is extracted and given to the model since the background and other portion of the image is unnecessary. For this, we use Face detection which is a computer vision technology that locates human faces in digital images. Haar Cascades Algorithm, also known Viola-Jones Algorithm is used for Face detection which was proposed by Paul Viola and Michael Jones in their 2001 paper.

Working of the algorithm:

This algorithm uses Haar features to extract objects.

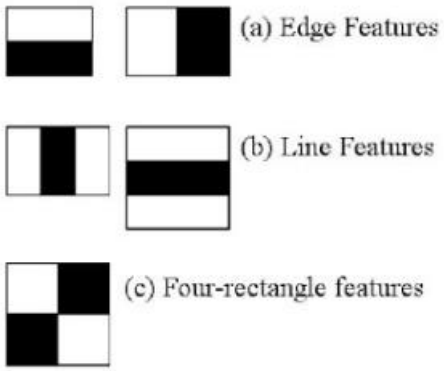

Fig. 2.2.2 - Haar Features.

The algorithm applies the features on windows of the image, gradually changing the window size after each turn. If a particular window is not classified as a face, that window is not processed in further steps. Instead of applying all the features, it makes use of cascade of classifiers concept. The features are grouped into different stages of classifiers and applied one-by-one. Only if a window passes a stage, the next stage is applied on that or else the window will be discarded and no longer considered as face region. A face region is the one which passes all the stages.

The openCV detectmultiscale() returns the starting coordinates, width and height of the detected face, using which the face region is extracted from the corresponding image and resized to a fixed size 100x100. We append all the resized images to a list and the corresponding image labels to another list. We convert both the lists to numpy arrays and we give these arrays to the Classification model.

The parameters used in detectmultiscale() method are image, scaleFactor and minNeighbors. The scaleFactor parameter specifies how much the image size is reduced at each image scale. The minNeighbor parameter specifies how many neighbors each candidate rectangle should have to retain in. This parameter affects the quality of the detected faces.

2.2.13 - The Classification Module

In this module, we build a Deep Learning Binary Classification model to classify the images as alert / drowsy. We used sequential model. In machine learning, classification refers to a predictive problem where a class label is predicted for a given example of input data. A model will use the training dataset and will calculate how to best map examples of input data to specific class labels. Binary classification refers to those classification tasks that have two class labels.

Convolutional Neural Networks, or CNNs, were designed to map image data to an output variable. The benefit of using CNNs is their ability to develop an internal representation of a two-dimensional image.

A convolution is essentially sliding a filter over the input. Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map

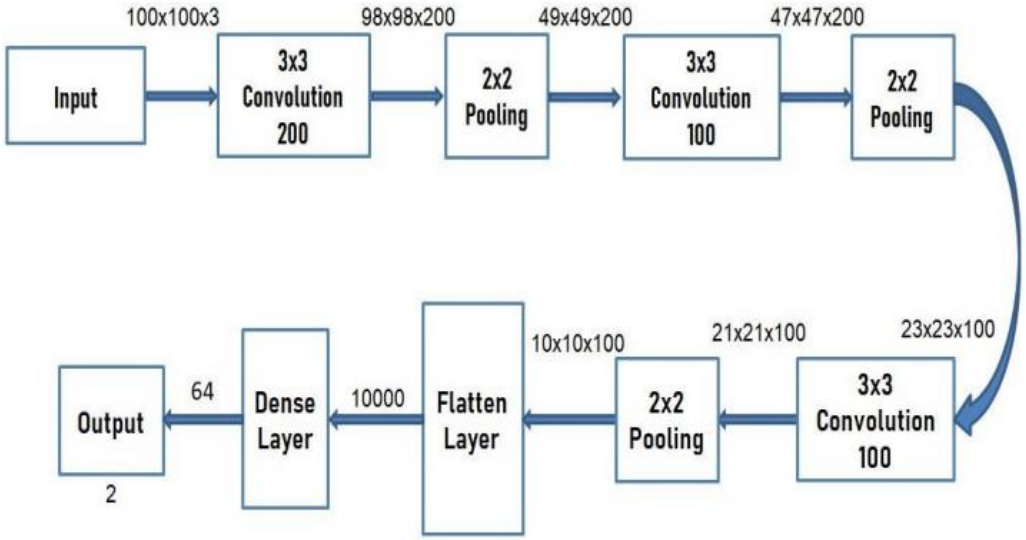

Fig. 2.2.3 CNN Architecture

We define the size of the input for the input layer as 100x100. We add the following layers to the model:

• A 2D convolutional layer with ReLU activation function. The hyper parameters for this layer are 200 (number of kernels) and (3x3) (each of size 3x3). The shape of each input image to this layer is (100x100) and the output shape is (98x98x200). Here, 200 are the number of kernels.

• A Max Pooling layer of pool size (2x2). The output shape of this layer is (49x49x200).

• A 2D convolutional layer with ReLU activation function. The hyper parameters for this layer are 100 (number of kernels) and (3x3) (each of size 3x3). The output shape of this layer is (47x47x100). Here, 100 are the number of kernels.

• A Max Pooling layer of pool size (2x2). The output shape of this layer is (23x23x100).

• A 2D convolutional layer with ReLU activation function. The hyper parameters for this layer are 100 (number of kernels) and (3x3) (each of size 3x3). The output shape of this layer is (21x21x100). Here, 100 are the number of kernels.

• A Max Pooling layer of pool size (2x2). The output shape of this layer is (10x10x100).

• A flatten layer which consists of 10000 neurons. The shape of input to this layer is (10x10) with 100 kernels. So, the output from this layer has 10000 neurons.

• A dense layer with 64 neurons with ReLU activation function.

• The output layer with 2 neurons as this is a binary classification model. It has sigmoid activation function.

Training:

We shuffle the data and split all the available images into train-data(80%) and testdata(20%). We run 20 epochs during training. While training the model, some part of the training data is used as validation data. We use 10% of the training data for validation. The validation data is used to evaluate the performance of the model at the end of each epoch but not used to train the model. We use checkpoints in our model to save the state of the model each time an improvement is observed during training. The checkpoint stores the weights of the model. The weights will be updated after each improvement and the best model will be saved when the training is completed.

Evaluation: After training the model, it's performance should be measured. There are different metrics to evaluate a Classification model like Loss, Accuracy, Precision and Recall. We use Matplotlib to plot the graphs of different metrics of the model.

Testing with test data:

Testing is used to evaluate the generalizing ability of the model by giving unseen data to it. The data is previously split into training data and test data. The 20% of the data which is not known to the model is given to the model and the model will predict the classes for the test data. We evaluate the performance of the model by finding loss and accuracy with the test data. We also find the Precision and Recall values for the model.

2.2.14 - Predicting the images captured from the camera

After the model is trained with the given dataset, we can use the model to predict the class of the images which are captured from the camera.

We use OpenCV to capture the images from the camera. We continuously capture image frames from the camera. The same preprocessing steps which are applied on the dataset are applied on each frame captured i.e., detecting the face from the image

frame, extract the Region of Interest and then resizing the Region of Interest to a fixed size (100x100). Then we convert the images into array format to give as input to the model. Then, we can give a set of images to the trained CNN Classification model to predict the labels for the images. If the driver is feeling drowsy, a voice alert is generated. An audio file is played using the playsound module in python.

Chapter 3- System Design

PROPOSED METHODOLOGY

3.1. Existing System

The existing system used Support Vector Machine (SVM) for classifying the face (if the eyes ae closed or not) as drowsy or not drowsy. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. The objective of SVM is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes.

3.2. Proposed System

In this system, instead of Support Vector Machine (SVM) we use a Classification Model based on Convolutional Neural Networks (CNN). Deep Learning is concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. Convolutional Neural Networks are a type of Artificial Neural Networks which are widely used for image classification and even Multi-Class classification of images. Convolution layers in a CNN consist of a set of learnable filters. During forward propagation, we slide each filter across the whole input step by step where each step is called stride. We propose CNN since the accuracy of the system is improved by using a CNN. The driver's face is continuously captured using a camera. The image frames are extracted and by face detection, the face of the driver is detected. A classification model is built based on CNN to classify the face as drowsy / not drowsy.

3.3. Proposed Techniques

3.3.1. Artificial Intelligence (AI)

Definition: Artificial Intelligence (AI) is the study and creation of computer systems that can perceive, reason and act. The primary aim of Al is to produce intelligent machines. The intelligence should be exhibited by thinking, making decisions, solving problems, more importantly by learning. Al is an interdisciplinary field that requires knowledge in computer science, linguistics, psychology, biology, philosophy and so on for serious research.2 According to the father of Artificial Intelligence, John McCarthy, it is the science and engineering of making intelligent machines, especially intelligent computer programs. It is a way of Making a Computer, a Computer-Controlled Robot, or a Software Think Intelligently in the similar manner the intelligent humans think

Artificial intelligence (AI), the ability of a digital computer or computercontrolled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience. Since the development of the digital computer in the 1940s, it has been demonstrated that computers can be programmed to carry out very complex tasks—as, for example, discovering proofs for mathematical theorems or playing chess— with great proficiency. Still, despite continuing advances in computer processing speed and memory capacity, there are as yet no programs that can match human flexibility over wider domains or in tasks requiring much everyday knowledge. On the other hand, some programs have attained the performance levels of human experts and professionals in performing certain specific tasks, so that artificial intelligence in this limited sense is found in applications as diverse as medical diagnosis, computer search engines, and voice or handwriting recognition.

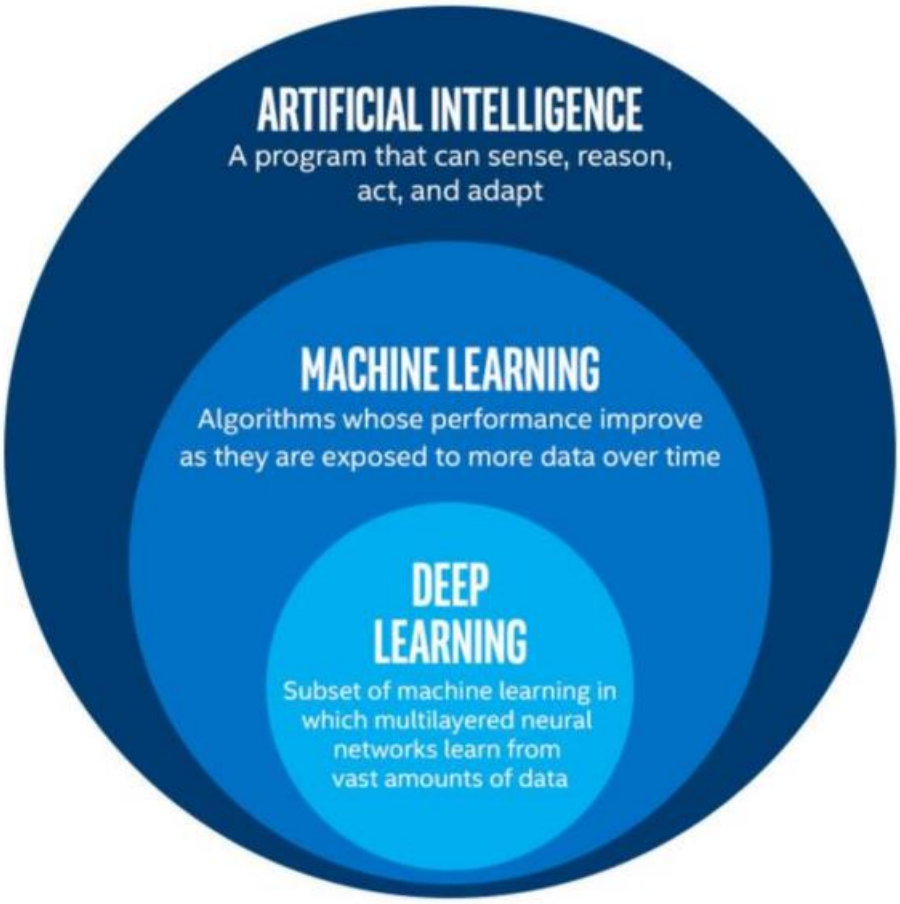

Fig. 3.3.1.1. Artificial Intelligence

3.3.2. Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Machine Learning is defined as the study of computer programs that leverage algorithms and statistical models to learn through inference and patterns without being explicitly programed. Machine Learning field has undergone significant developments in the last decade. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

There are also some types of machine learning algorithms that are used in very specific use-cases, but three main methods used today are: Supervised Machine Learning Algorithm Unsupervised Machine Learning Algorithm Reinforcement Machine Learning Algorithm

Among these, the type of machine learning algorithm we used in our system is supervised machine learning algorithm.

1. Supervised machine learning

This can apply what has been learned in the past to new data using labelled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

2. Unsupervised machine learning

It holds the advantage of being able to work with unlabeled data. This means that human labor is not required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program. In supervised learning, the labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work off of, resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner, with no input required from human beings. The creation of these hidden structures is what makes unsupervised learning algorithms versatile. Instead of a defined and set problem statement, unsupervised learning algorithms can adapt to the data by dynamically changing hidden structures. This offers more postdeployment development than supervised learning algorithms.

3. Reinforcement Machine Learning

It directly takes inspiration from how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged or 'reinforced', and nonfavorable outputs are discouraged or 'punished'. Based on the psychological concept of conditioning, reinforcement learning works by putting the algorithm in a work environment with an interpreter and a reward system.

In every iteration of the algorithm, the output result is given to the interpreter, which decides whether the outcome is favorable or not. In case of the program finding the correct solution, the interpreter reinforces the solution by providing a reward to the algorithm. If the outcome is not favorable, the algorithm is forced to reiterate until it finds a better result. In most cases, the reward system is directly tied to the effectiveness of the result. In typical reinforcement learning use-cases, such as finding the shortest route between two points on a map, the solution is not an absolute value. Instead, it takes on a score of effectiveness, expressed in a percentage value. The higher this percentage value is, the more reward is given to the algorithm. Thus, the program is trained to give the best possible solution for the best possible reward.

3.3.3. Deep Learning

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

It is a form of machine learning, with functions that operate in a nonlinear decision-making process. Deep learning occurs when decisions are made on unstructured data without supervision. Object recognition, speech recognition, and language translation are some of the tasks performed through deep learning.

Deep learning is an AI function that mimics the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions.

Deep learning AI is able to learn without human supervision, drawing from data that is both unstructured and unlabelled. It is a form of machine learning, can be used to help detect fraud or money laundering, among other functions.

3.3.4. Neural Networks

Neural networks are artificial systems that were inspired by biological neural networks. These systems learn to perform tasks by being exposed to various datasets and examples without any task-specific rules. The idea is that the system generates identifying characteristics from the data they have been passed without being programmed with a preprogrammed understanding of these datasets. Components of a typical neural network involve neurons, connections, weights, biases, propagation function, and a learning rule. Neurons will receive an input from predecessor neurons that have an activation, threshold, an activation function f, and an output function. Connections consist of connections, weights and biases which rules how neuron i transfers output to neuron j. Propagation computes the input and outputs the output and sums the predecessor neurons function with the weight. The learning rule modifies the weights and thresholds of the variables in the network. Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Networks in a system. A layer can have only a dozen units or millions of units as this depends on the complexity of the system. Commonly, Artificial Neural Network has an input layer, output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided. In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.

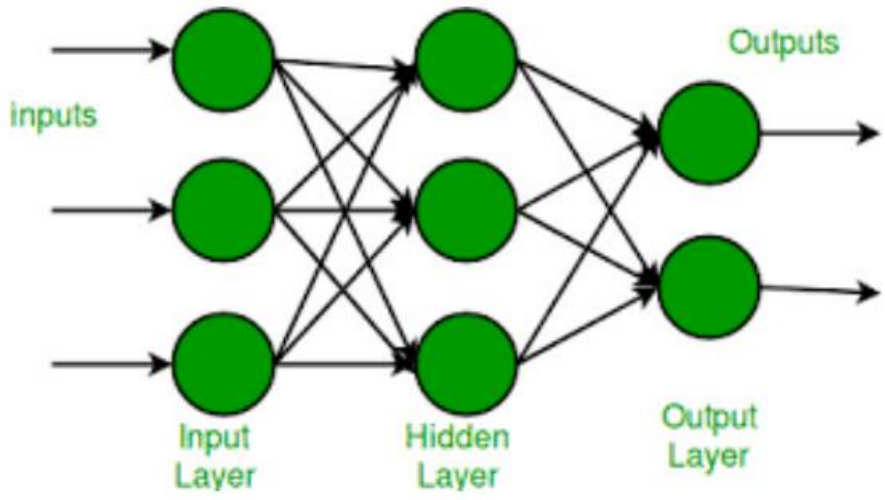

Fig. 3.3.4.1. Neural Networks

Types:

1. Feedforward Neural Network

The feedforward neural network is one of the most basic artificial neural networks. In this ANN, the data or the input provided ravels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist. So the feedforward neural network has a front propagated wave only and usually does not have backpropagation.

2. Recurrent Neural Network

The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer. The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed. After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computations.

3. Convolutional Neural Network

A Convolutional neural network has some similarities to the feed-forward neural network, where the connections between units have weights that determine the influence of one unit on another unit. But a CNN has one or more than one convolutional layers that use a convolution operation on the input and then pass the result obtained in the form of output to the next layer. CNN has applications in speech and image processing which is particularly useful in computer vision.

4. Modular Neural Network

A Modular Neural Network contains a collection of different neural networks that work independently towards obtaining the output with no interaction between them. Each of the different neural networks performs a different sub-task by obtaining unique inputs compared to other networks. The advantage of this modular neural network is that it breaks down a large and complex computational process into smaller components, thus decreasing its complexity while still obtaining the required output.

5. Radial basis function Neural Network

Radial basis functions are those functions that consider the distance of a point concerning the center. RBF functions have two layers. In the first layer, the input is mapped into all the Radial basis functions in the hidden layer and then the output layer computes the output in the next step. Radial basis function nets are normally used to model the data that represents any underlying trend or function.

3.3.5. Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It's high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's.

simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective. It's easy to learn syntax and portability capability makes it popular these days.

The followings facts give us the introduction to Python:

1.Python was developed by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands.

2.It was written as the successor of programming language named 'ABC'.

3.It's first version was released in 1991.

4.The name Python was picked by Guido van Rossum from a TV show named Monty Python's Flying Circus.

5.It is an open source programming language which means that we can freely download it and use it to develop programs. It can be downloaded from www.python.org.

6.Python programming language is having the features of Java and C both. It is having the elegant 'C' code and on the other hand, it is having classes and objects like Java for object-oriented programming.

7.It is an interpreted language, which means the source code of Python program would be first converted into bytecode and then executed by Python virtual machine.

Python for Data Science

Whether you're doing straightforward data analysis or full-on data science, you'd be hardpressed to find a better suite of tools than those in Python.The Pandas library is a quantum-leap improvement over the dusty Excel spreadsheets in which financial analysis was done for so long. If Pandas isn't fast enough for you, most of the basic vector operations can be one with NumPy. Numpy also offers the ability to do linear algebra, scientific computing, and a host of other highly technical things. It is, therefore, a great tool to learn how to use well. Python is the fifth most important language as well as most popular language for Machine learning and data science. The following are the features of Python that makes it the preferred choice of language for data science:

1.Extensive set of packages

2.Easy prototyping

3.Collaboration feature

4.One language for many domains

Artificial Intelligence and Machine Learning

Python community has developed many modules to help programmers implement machine learning. Artificial intelligence and machine learning have become buzzwords these days, but the truth is that it all comes down to algorithms, code, and logic. Given the scope and power of Python, it's no surprise that some truly world-class tools exist for generating intelligent behavior in Python. Arguably the most popular is the ubiquitous machine learning library, Scikit-Learn. Speaking from experience, Sklearn makes the process of building everything from classifiers to regression models orders of magnitude simpler than it otherwise would be.If neural networks are more your jam, there's always TensorFlow. Adding in the new Keras API, building a state-of-the-art neural network is easier than it has ever been.

3.3.6. Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

The Jupyter notebook has two components. Users input programming code or text in rectangular cells in a front-end web page. The browser then passes that code to a back-end 'kernel', which runs the code and returns the results (see our example at go.nature.com/2yqq7ak). By Pérez's count, more than 100 Jupyter kernels have been created, supporting dozens of programming languages. Normally, each notebook can run only one kernel and one language, but workarounds exist.

Importantly, the kernels need not reside on the user's computer. When future users of the LSST use Jupyter notebooks to analyse their data, the code will be running on a supercomputer in Illinois, providing computational muscle no desktop PC could match. Notebooks can also run in the cloud. Google's Colaboratory project, for instance, provides a Google-themed front-end to the Jupyter notebook. It enables users to collaborate and run code that exploits Google's cloud resources — such as graphical processing units — and to save their documents on Google Drive.

Two additional tools have enhanced Jupyter's usability. One is JupyterHub, a service that allows institutions to provide Jupyter notebooks to large pools of users. The IT team at the University of California, Berkeley, where Pérez is a faculty member, has deployed one such hub, which Pérez uses to ensure that all students on his data-science course have identical computing environments. "We cannot possibly manage IT support for 800 students, helping them debug why the installation on their laptop is not working; that's simply infeasible," he says

The other development is Binder, an open-source service that allows users to use Jupyter notebooks on GitHub in a web browser without having to install the software or any programming libraries. Users can also execute Jupyter notebooks on the Google cloud by inserting https://colab.research.google.com/github before the URL of a notebook on GitHub, or using the commercial service Code Ocean. In September, Code Ocean rolled out a new user interface for its cloud-based code-sharing and code-execution service, also based on Jupyter.

3.3.7. Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

The term "Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified

by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.

There are two main parts to CNN architecture:

• A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction

• A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below

3.3.7.1. Convolutional Layer

Convolutional layers are the major building blocks used in convolutional neural networks. This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM). If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input image allows the filter an opportunity to discover that feature anywhere in the image

.

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The Convolution Operation We analyze the influence of nearby pixels by using something called a filter. We move this across the image from top left to bottom right. When building the network, we randomly specify values for the filters, which then continuously update themselves as the network is trained. After the filters have passed over the image, a feature map is generated for each filter. These are then taken through an activation function, which decides whether a certain feature is present at a given location in the image.

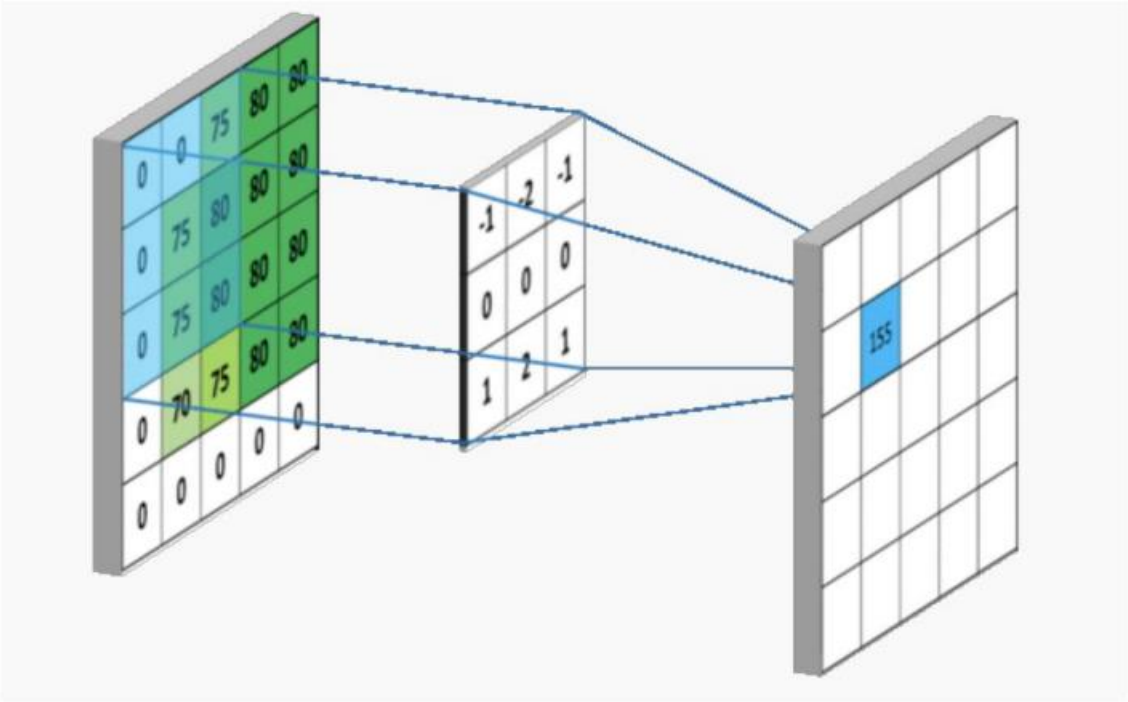

Fig. 3.3.7.1. Convolution Operation

We have colour images which have 3 channels i.e., Red, Green and Blue. Each filter is convolved with the entirety of the 3D input cube but generates a 2D feature map. Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter. Convolving the image with a filter produces a feature map that highlights the presence of a given feature in the image.

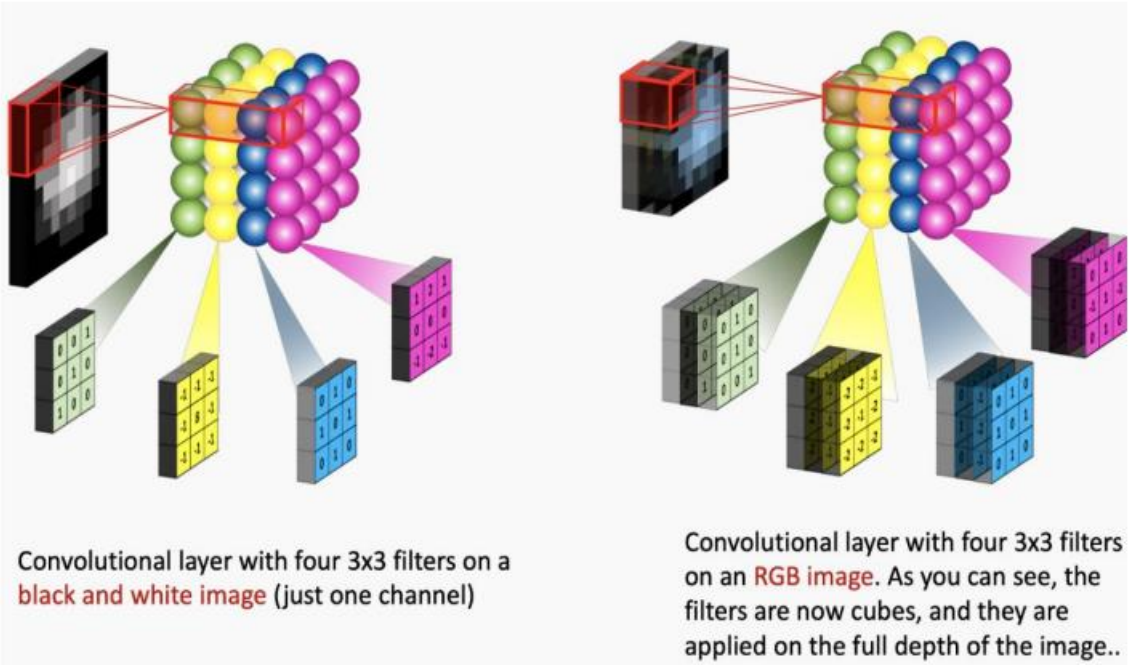

Fig. 3.3.7.2. Convolution with RGB images

Stride

Stride is the number of pixels shifts over the input matrix. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2

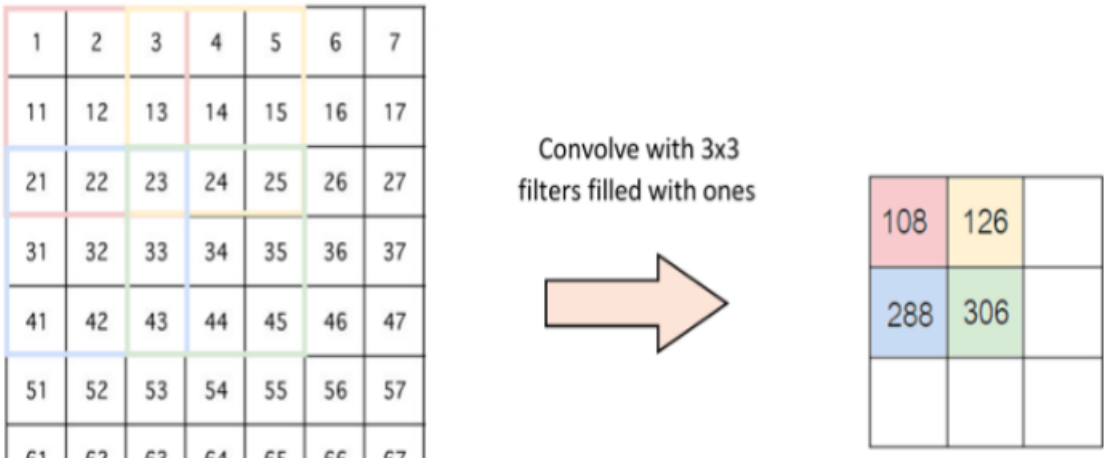

Fig. 3.3.7.3. Stride

3.3.7.2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

3.3.7.2.1. Max Pooling

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

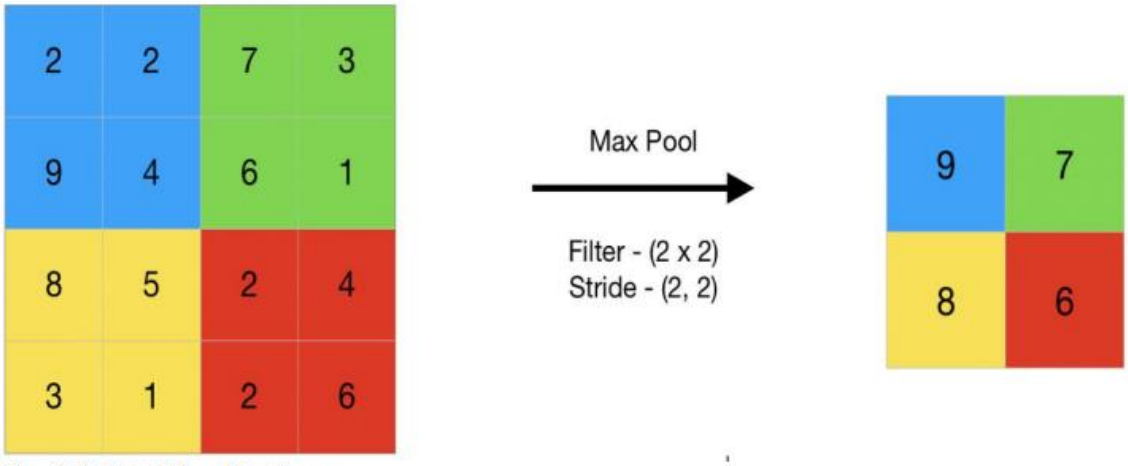

Fig. 3.3.7.4. Max Pooling

3.3.7.2.2. Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

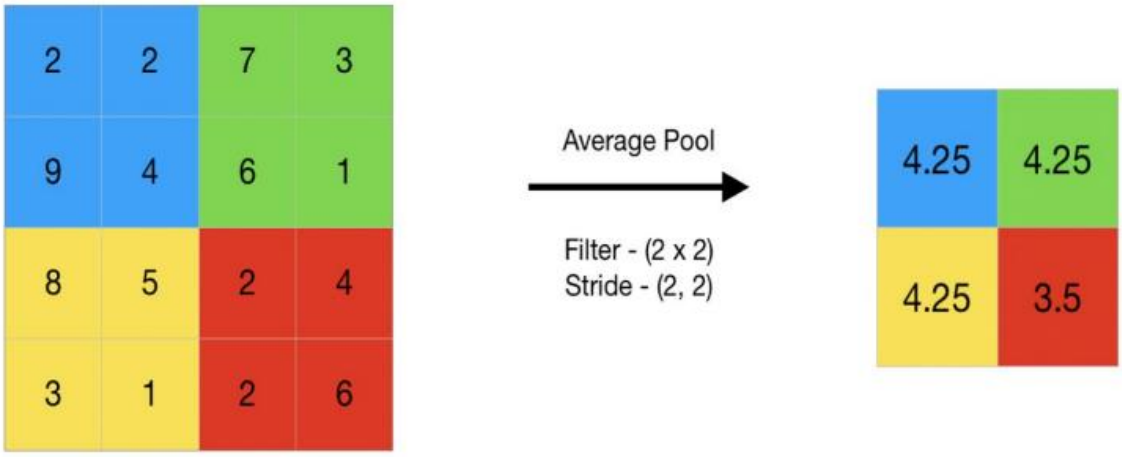

Fig. 3.3.7.5. Average Pooling

3.3.7.2.3. Global Pooling

Global pooling reduces each channel in the feature map to a single value. Thus, an nh x nw x nc feature map is reduced to 1 x 1 x nc feature map. This is equivalent to using a filter of dimensions nh x nw i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling

3.3.7.3. Fully Connected layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

3.3.7.4. Dropout

When all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

3.3.8. Back Propagation

Backpropagation is an algorithm commonly used to train neural networks. When the neural network is initialized, weights are set for its individual elements, called neurons. Inputs are loaded, they are passed through the network of neurons, and the network provides an output for each one, given the initial weights.

Backpropagation helps to adjust the weights of the neurons so that the result comes closer and closer to the known true result. Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptron's to multilayer feedforward neural networks. The "backwards" part of the name stems from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately.

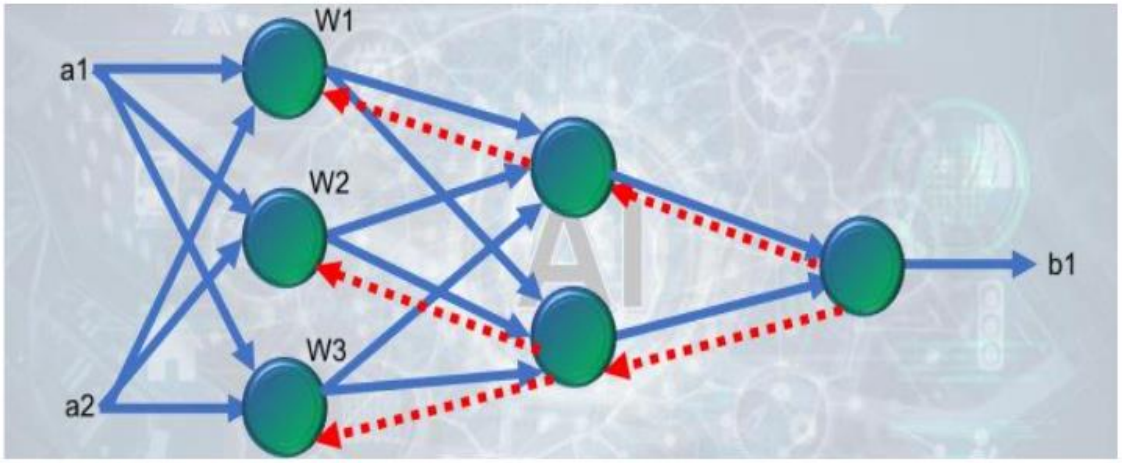

Fig. 3.3.8.1. Back Propagation

Backpropagation's popularity has experienced a recent resurgence given the widespread adoption of deep neural networks for image recognition and speech recognition. It is considered an efficient algorithm, and modern implementations take advantage of specialized GPUs to further improve performance.

3.3.9. Activation Functions

One of the most important parameters of the Neural Networks is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.

It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions has a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred for a multi-class classification, generally softmax us used.

Some of the Activation Functions are:

3.3.9.1. Binary Step Function

If the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated

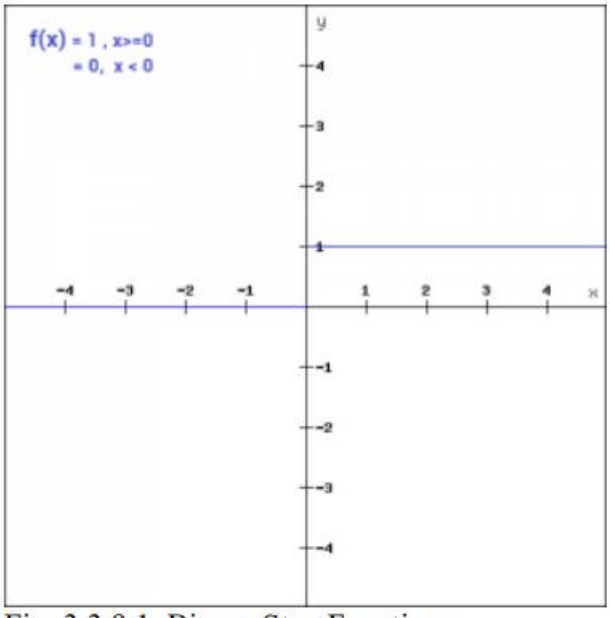

Fig. 3.3.9.1. Binary Step Function

3.3.9.2. Linear Function

Here the activation is proportional to the input. The variable ‘a’ in this case can be any constant value.

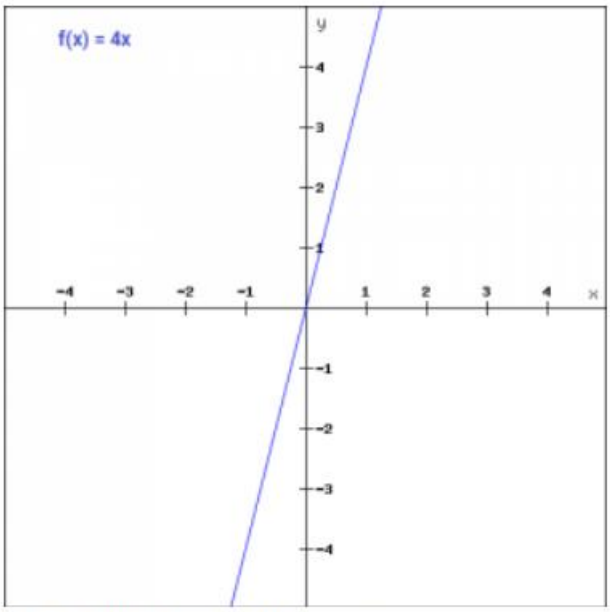

Fig. 3.3.9.2. Linear Function

3.3.9.3. Sigmoid

It is one of the most widely used non-linear activation function. Sigmoid transforms the values between the range 0 and 1.

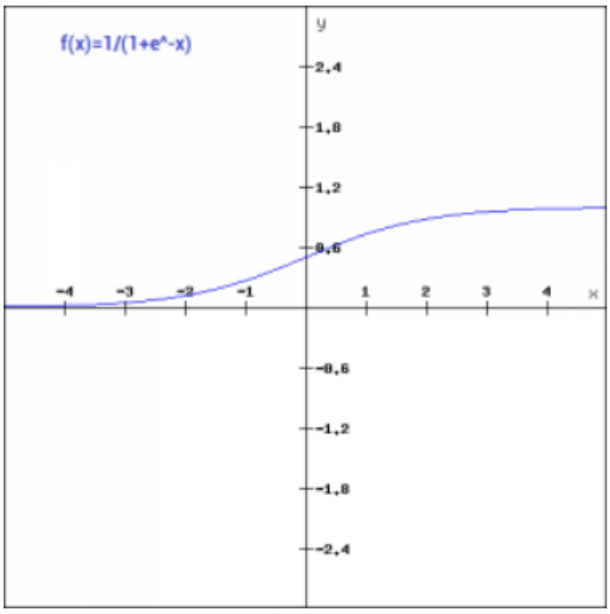

Fig. 3.3.9.3. Sigmoid Function

3.3.9.4. Tanh

The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of values in this case is from -1 t

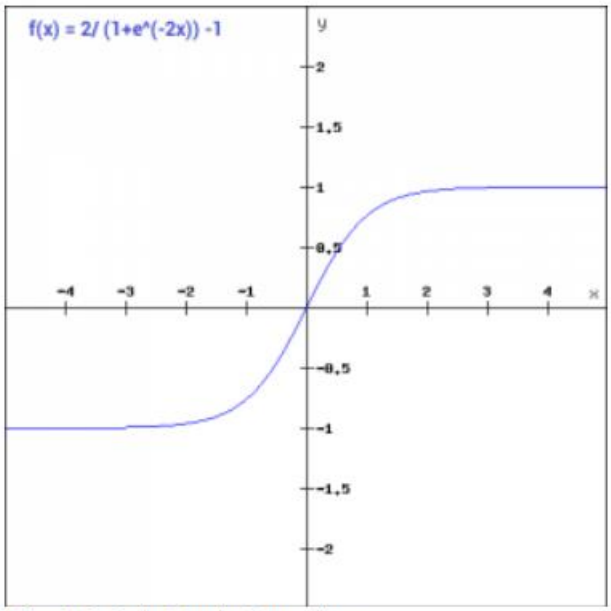

Fig. 3.3.9.4. Tanh Function

3.3.9.5. ReLU

The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The neurons will only be deactivated if the output of the linear transformation is less than 0.

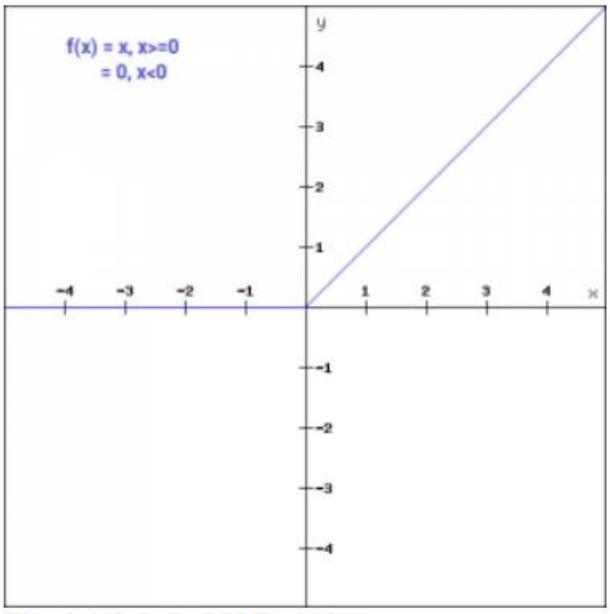

Fig. 3.3.9.5. ReLU Function

3.3.9.6. Softmax

Softmax function is often described as a combination of multiple sigmoids. We know that sigmoid returns values between 0 and 1, which can be treated as probabilities of a data point belonging to a particular class. The softmax function can be used for multiclass classification problems. This function returns the probability for a datapoint belonging to each individual class.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

3.3.10. Training

Deep learning neural networks learn a mapping function from inputs to outputs. This is achieved by updating the weights of the network in response to the errors the model makes on the training dataset. Updates are made to continually reduce this error until either a good enough model is found or the learning process gets stuck and stops.

The stochastic gradient descent algorithm is used to solve the optimization problem where model parameters are updated each iteration using the backpropagation algorithm.

A neural network model uses the examples to learn how to map specific sets of input variables to the output variable. It must do this in such a way that this mapping works well for the training dataset, but also works well on new examples not seen by the model during training. This ability to work well on specific examples and new examples is called the ability of the model to generalize.

• Loss Function: The function used to estimate the performance of a model with a specific set of weights on examples from the training dataset.

• Weight Initialization: The procedure by which the initial small random values are assigned to model weights at the beginning of the training process.

• Batch Size: The number of examples used to estimate the error gradient before updating the model parameters.

• Learning Rate: The amount that each model parameter is updated per cycle of the learning algorithm.

• Epochs: The number of complete passes through the training dataset before the training process is terminated.

Train Dataset:

The sample of data used to fit the model. The part of data we use to train our model. This is the data which your model actually sees (both input and output) and learn from.

3.3.11. Testing

The usage of the word “testing” in relation to Machine Learning models is primarily used for testing the model performance in terms of accuracy/precision of the model. It can be noted that the word, "testing" means different for conventional software development and Machine Learning models development.

The goal of ML testing:

Quality assurance is required to make sure that the software system works according to the requirements. Were all the features implemented as agreed? Does the program behave as expected? All the parameters that you test the program against should be stated in the technical specification document.

Moreover, testing has the power to point out all the defects and flaws during development. We don't want your clients to encounter bugs after it is released and come to you waving their fists. Different kinds of testing allow us to catch bugs that are visible only during runtime.

However, in machine learning, a programmer usually inputs the data and the desired behavior, and the logic is elaborated by the machine. This is especially true for deep learning. Therefore, the purpose of machine learning testing is, first of all, to ensure that this learned logic will remain consistent, no matter how many times we call the program. First of all, we split the database into three non-overlapping sets. You use a training set to train the model. Then, to evaluate the performance of the model, you use two sets of data:

Validation Dataset:

Having only a training set and a testing set is not enough if you do many rounds of hyper parameter-tuning (which is always). And that can result in overfitting. To avoid that, you can select a small validation data set to evaluate a model. Only after you get maximum accuracy on the validation set, you make the testing set come into the game.

Test Dataset:

Our model might fit the training dataset perfectly well. In order to assure that it will do equally well in real-life, you select samples for a testing set from your training set — examples that the machine hasn't seen before. It is important to remain unbiased during selection and draw samples at random. Also, you should not use the same set many times to avoid training on your test data. Your test set should be large enough to provide statistically meaningful results and be representative of the data set as a whole.

Chapter 4 – Project Management

4.1 Project Planning and scheduling

1.Define the Project Scope:

- Clearly define the goals and objectives of the project, such as developing a machine learning model for drowsiness detection using various physiological signals.
- Specify the target platform or system where the model will be deployed, such as a standalone device or integrated into existing systems.

2.Identify Stakeholders:

- Identify the key stakeholders involved in the project, such as researchers, developers, project managers, and end-users.
- Determine their roles, responsibilities, and expectations.

3.Gather Requirements:

- Conduct a thorough analysis of the requirements for the drowsiness detection system, including the necessary physiological signals, data collection methods, and hardware/software requirements.
- Define the desired accuracy, response time, and user interface requirements.

4.Data Collection and Annotation:

- Plan the collection of relevant data, such as electroencephalography (EEG), electrooculography (EOG), or other physiological signals.
- Develop a data annotation process to label the collected data for training the machine learning model.

5.Select Machine Learning Algorithms:

- Research and select appropriate machine learning algorithms suitable for drowsiness detection, such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), or Support Vector Machines (SVM).
- Consider factors like model complexity, accuracy, and computational resources required.

6.Develop the Machine Learning Model:

- Implement the selected algorithms using a programming language such as Python and relevant libraries (e.g., TensorFlow, PyTorch).
- Train and optimize the model using the annotated data.
- Validate the model using appropriate evaluation metrics and techniques, such as cross-validation or holdout validation.

7.Build the Drowsiness Detection System:

- Develop the necessary software components to integrate the trained model into a functional system.
- Implement the required user interface and interaction components.
- Integrate the system with the selected hardware, such as sensors for data collection.

8.Test and Validate:

- Conduct rigorous testing of the drowsiness detection system to ensure its accuracy, reliability, and robustness.
- Perform both functional and non-functional testing, including unit testing, integration testing, and performance testing.
- Validate the system's performance against predefined metrics and requirements.

9.Deployment and Rollout:

- Plan the deployment strategy for the drowsiness detection system, considering factors like user training, hardware setup, and maintenance procedures.
- Develop user documentation and provide necessary training to end-users and system administrators.
- Ensure proper integration with the target platform and system.

10.Project Management and Monitoring:

- Establish a project management framework to track progress, manage resources, and monitor milestones.
- Use project management tools and techniques, such as Gantt charts, task management systems, and regular status meetings.
- Continuously monitor the system's performance post-deployment and address any issues or improvements.

11.Documentation and Reporting:

- Document all project activities, including requirements, design decisions, implementation details, and test results.
- Prepare periodic progress reports for stakeholders, highlighting key achievements, challenges, and future plans.
- Document the final project outcomes, including the drowsiness detection system's performance, limitations, and recommendations for future enhancements.
- Remember to adjust the timeline and resources based on the specific requirements of your project. Project planning and scheduling may vary depending on factors like team size, complexity, and available resources.

4.1.1 Project Development Approach

1.Problem Understanding and Data Collection:

- Understand the problem of drowsiness detection and its significance.
- Identify the relevant physiological signals to be used for drowsiness detection, such as EEG, EOG, or other sensor data.
- Collect a diverse and representative dataset of labeled data, including instances of drowsy and non-drowsy states.

2.Data Preprocessing:

- Clean the collected data by removing noise, artifacts, or outliers.
- Normalize or standardize the data to ensure consistency and comparability across different features.
- Perform feature engineering to extract meaningful and informative features from the raw data, such as frequency or time-domain features.

3.Model Selection:

- Research and select appropriate machine learning algorithms for drowsiness detection, considering factors like accuracy, interpretability, and computational resources.
- Commonly used algorithms include Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Support Vector Machines (SVM), or Ensemble methods.

4.Model Training:

- Split the preprocessed dataset into training, validation, and test sets.

- Train the selected machine learning model using the training set.
- Optimize the model hyperparameters through techniques like grid search, random search, or Bayesian optimization.
- Validate the model's performance using the validation set, adjusting the model and hyperparameters as necessary.

5.Model Evaluation:

- Evaluate the trained model's performance on the test set using appropriate evaluation metrics, such as accuracy, precision, recall, or F1 score.
- Perform a comprehensive analysis of the model's performance, considering false positives, false negatives, and the impact of different thresholds.

6.Deployment and Integration:

- Integrate the trained model into a drowsiness detection system, considering factors like hardware requirements, real-time performance, and user interface.
- Develop the necessary software components to handle data acquisition, processing, and inference based on the trained model.
- Ensure the system is compatible with the target platform and devices.

7.Testing and Validation:

- Conduct rigorous testing of the drowsiness detection system to ensure its accuracy, reliability, and robustness.
- Perform both functional and non-functional testing, including unit testing, integration testing, and performance testing.
- Validate the system's performance against predefined metrics and requirements, considering real-world scenarios and different user contexts.

8.Continuous Improvement and Maintenance:

- Monitor the performance of the deployed drowsiness detection system and gather feedback from end-users.
- Continuously improve the system's performance and accuracy by retraining the model with new data or fine-tuning the existing model.
- Regularly update the system to address any issues, incorporate new features, or adapt to changing user needs.
- It's important to note that the development approach may vary depending on the specific requirements, available resources, and constraints of the project. Additionally, collaboration between domain experts, data scientists, and software engineers is crucial for successful development and deployment.

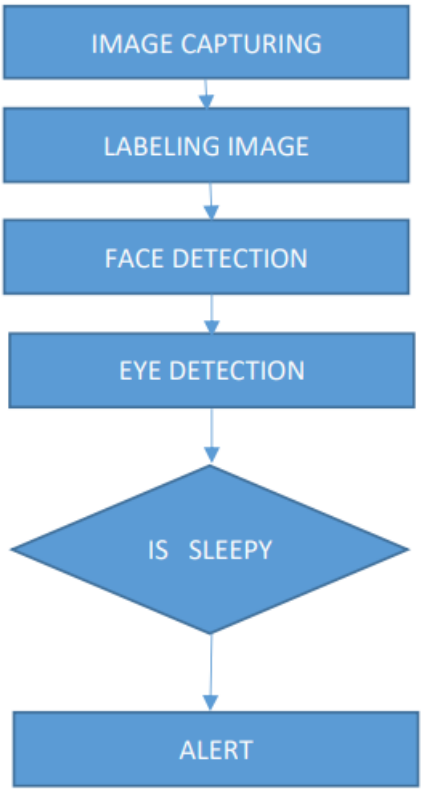

Fig 4.1.1 Possible Approach

4.1.2 Project Plan

1.Project Initiation Phase:

- Define project goals, objectives, and scope.
- Identify key stakeholders and their roles.
- Establish project team and communication channels.
- Conduct initial research on drowsiness detection methods and available datasets.
- Develop a project charter outlining the project's purpose, objectives, and success criteria.

2.Requirements Gathering Phase:

- Define functional and non-functional requirements for the drowsiness detection system.
- Identify the target platform and hardware/software specifications.
- Determine the required physiological signals for drowsiness detection.
- Specify desired accuracy, real-time response, and user interface requirements.
- Create a requirements document to serve as a reference throughout the project.

3.Data Collection and Annotation Phase:

- Identify and collect a representative dataset of physiological signals (e.g., EEG, EOG) for drowsiness and non-drowsiness states.
- Annotate the collected data to label instances of drowsiness and non-drowsiness.
- Perform data preprocessing tasks, such as cleaning, filtering, and feature extraction.

4.Model Development Phase:

- Research and select appropriate machine learning algorithms for drowsiness detection (e.g., CNN, RNN).
- Split the annotated dataset into training, validation, and test sets.
- Develop and train the selected machine learning model using the training data.
- Optimize the model's hyperparameters through techniques like cross-validation or grid search.
- Evaluate the model's performance using appropriate metrics and validation data.

5.System Development Phase:

- Develop software components for data acquisition from sensors or input devices.
- Implement data processing and feature extraction algorithms.
- Integrate the trained machine learning model into the system.
- Build a user-friendly interface for system interaction and feedback.
- Conduct unit testing and integration testing of the system components.

6.System Testing and Validation Phase:

- Perform comprehensive testing of the drowsiness detection system.
- Conduct functional testing to ensure all requirements are met.
- Validate the system's performance against predefined metrics and user scenarios.

- Fine-tune the model and system components based on test results and feedback.
- Document any issues or bugs encountered during testing and propose resolutions.

7.Deployment and Rollout Phase:

- Prepare the system for deployment on the target platform.
- Establish necessary hardware configurations and software dependencies.
- Create installation and configuration instructions.
- Provide user training and support materials.
- Deploy the drowsiness detection system in the target environment.

8.Monitoring and Maintenance Phase:

- Monitor the system's performance and accuracy in real-world conditions.
- Gather feedback from end-users and stakeholders for further improvements.
- Regularly update and maintain the system, incorporating bug fixes and enhancements.
- Conduct periodic model retraining with new data to improve performance.
- Document the system's maintenance procedures and support channels.

9.Project Closure Phase:

- Conduct a project review to evaluate the overall success and lessons learned.
- Document project outcomes, including system performance, limitations, and future recommendations.
- Archive project artifacts, including codebase, documentation, and datasets.
- Share project findings and outcomes with relevant stakeholders.
- Celebrate project completion and recognize team contributions.

4.2 Risk Management

4.2.1 Risk Identification and Risk Analysis

1.Insufficient or Inaccurate Training Data:

- Risk: The availability of insufficient or inaccurate training data can impact the performance and generalization ability of the machine learning model.
- Mitigation: Ensure a diverse and representative dataset for training the model. Perform data quality checks, data augmentation, or seek expert input to improve the dataset quality.

2.Model Overfitting:

- Risk: Overfitting occurs when the model learns the training data too well but fails to generalize to unseen data, resulting in poor performance.
- Mitigation: Use techniques like cross-validation, regularization, and early stopping to prevent overfitting. Consider ensemble methods or transfer learning to improve generalization.

3.Computational Resource Limitations:

- Risk: Training and inference processes may require significant computational resources, such as memory and processing power.
- Mitigation: Optimize the model architecture and hyperparameters to reduce computational requirements. Consider distributed training or cloud-based solutions to leverage more resources if necessary.

4.System Integration Challenges:

- Risk: Integrating the drowsiness detection system with existing hardware or software systems may present compatibility or interoperability challenges.
- Mitigation: Conduct thorough compatibility testing and collaborate with relevant stakeholders to address integration issues. Use standardized protocols and interfaces for seamless integration.

5.False Positives or False Negatives:

- Risk: The model may produce false positives (incorrectly detecting drowsiness) or false negatives (failing to detect drowsiness).
- Mitigation: Continuously evaluate and fine-tune the model using appropriate evaluation metrics. Adjust decision thresholds based on domain expertise and feedback from end-users.

6.Ethical and Legal Considerations:

- Risk: Drowsiness detection systems may collect sensitive data, raise privacy concerns, or have legal implications.
- Mitigation: Adhere to ethical guidelines and legal regulations regarding data privacy and security. Implement proper anonymization and encryption techniques. Obtain informed consent from data subjects.

7.System Reliability and Safety:

- Risk: Inaccurate or unreliable drowsiness detection can have serious safety implications, especially in applications like driver monitoring systems.
- Mitigation: Conduct thorough testing and validation to ensure the system's reliability and safety. Implement fail-safe mechanisms and redundancies where necessary.

8.Project Delays or Resource Constraints:

- Risk: Insufficient resources, unexpected delays, or unforeseen circumstances may impact the project timeline and deliverables.
- Mitigation: Develop a realistic project plan with contingency buffers. Continuously monitor progress and adjust resources or priorities as needed. Communicate and manage expectations with stakeholders.

9.Model Bias and Fairness:

- Risk: Machine learning models may exhibit biases or unfairness, leading to differential performance or discrimination across different user groups.
- Mitigation: Evaluate the model for biases and fairness using appropriate metrics. Regularly assess and address any biases or disparities to ensure equitable performance for all users.

10.Model Interpretability and Explainability:

- Risk: Complex machine learning models may lack interpretability, making it challenging to understand and explain their decisions.
- Mitigation: Utilize techniques like feature importance analysis, model visualization, or surrogate models to enhance interpretability. Balance model complexity with explainability requirements.

4.3 Estimation

4.3.1 Cost Analysis:

1.Data Collection and Annotation:

- Cost of acquiring or collecting the necessary physiological data for drowsiness detection, such as EEG or EOG signals.
- Expenses related to data annotation, including hiring annotators or using annotation services.

2.Infrastructure and Hardware:

- Cost of computing resources, including servers, GPUs, or cloud services, required for model training and inference.
- Hardware costs for sensors or devices used to capture physiological signals.

3.Software and Tools:

- Expenses related to software licenses, development tools, or libraries for data preprocessing, model development, and system implementation.

4.Human Resources:

- Salaries or wages for data scientists, machine learning engineers, software developers, and project managers involved in the project.
- Costs associated with recruiting, training, and retaining skilled personnel.

5.Model Development and Training:

- Time and effort required for data preprocessing, feature engineering, and model development.
- Costs associated with model training, including iterations, hyperparameter tuning, and optimization.

6.System Development and Integration:

- Costs related to developing software components for data acquisition, processing, and real-time inference.
- Expenses associated with integrating the drowsiness detection system with existing hardware or software infrastructure.

7.Testing and Validation:

- Costs for conducting thorough testing and validation of the system, including functional testing, performance testing, and user acceptance testing.
- Expenses related to data collection or simulation for testing different scenarios and user contexts.

8.Deployment and Maintenance:

- Costs associated with deploying the drowsiness detection system on the target platform or devices.
- Expenses for ongoing maintenance, bug fixes, and system updates.

9.Training and Support:

- Costs for training end-users, providing user documentation, and technical support.
- Expenses associated with user training materials, workshops, or webinars.

10.Ethical and Legal Considerations:

- Costs related to ensuring compliance with ethical guidelines and data privacy regulations.
- Expenses for legal consultations or audits regarding data privacy and security.
- It's important to note that the specific costs will vary depending on factors such as the project scale, complexity, and available resources. Creating a detailed budget that includes all the relevant cost factors and regularly tracking and monitoring expenses can help ensure the project stays within budget and allows for proper financial planning and decision-making.

Chapter 5 - Input Design

Input:

The input is an image which contains a human face. The following are some of the images from our dataset:

Fig 5.1 Open Eye Image

Fig 5.2 Closed Eye And Drowsy Eye

Detected Faces:

- The following are the images after performing face detection and resizing of the images:
- **Haar cascade is an algorithm** that can detect objects in images, irrespective of their scale in image and location. This algorithm is not so complex and can run in real-time.

Fig 5.3 Detected Only Face

Chapter 6- Output Design

Output:

If the driver is drowsy, an alert (voice) is generated as the output.

Results on test data:
Test loss: 0.37542635202407837
Test accuracy: 0.9086757898330688
Test precision: 0.6204379796981812
Test recall: 0.77625572681427

Results of frames captured from camera:

1.Frames classified as drowsy:
Here the input will be the continuous stream of video, if the driver seems to be detected as drowsy then the detected face will be highlighted using a red colored rectangle and system gives an alert. The alert will be in the form of a sound (beep sound). The aim is to make the driver aware that he/she is drowsy with that sound. The drowsiness is detected by using the well trained CNN model.

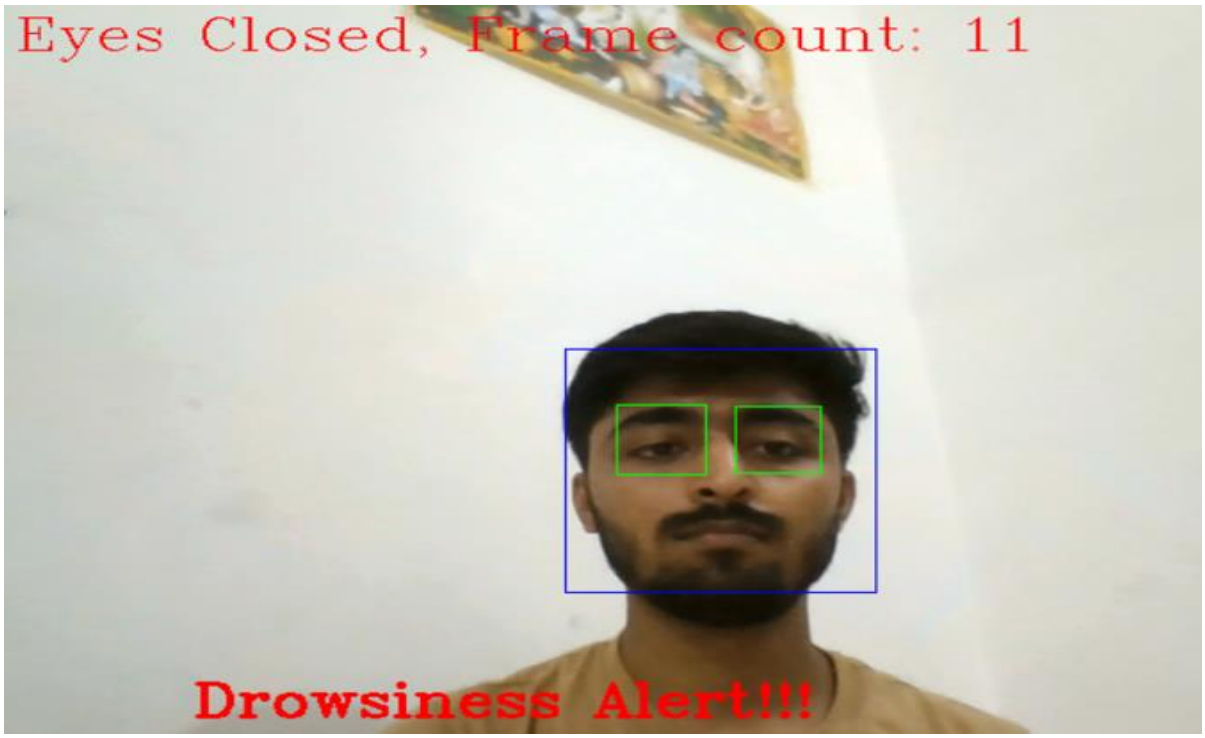

Fig 6.1 Drowsiness Alert

2.Frames classified as alert (not drowsy):

Here the input will be the continuous stream of video, if the driver seems to be detected as not drowsy then the detected face will be highlighted use a green colored rectangle and no alert will be generated. The drowsiness is detected by using the well trained CNN model.

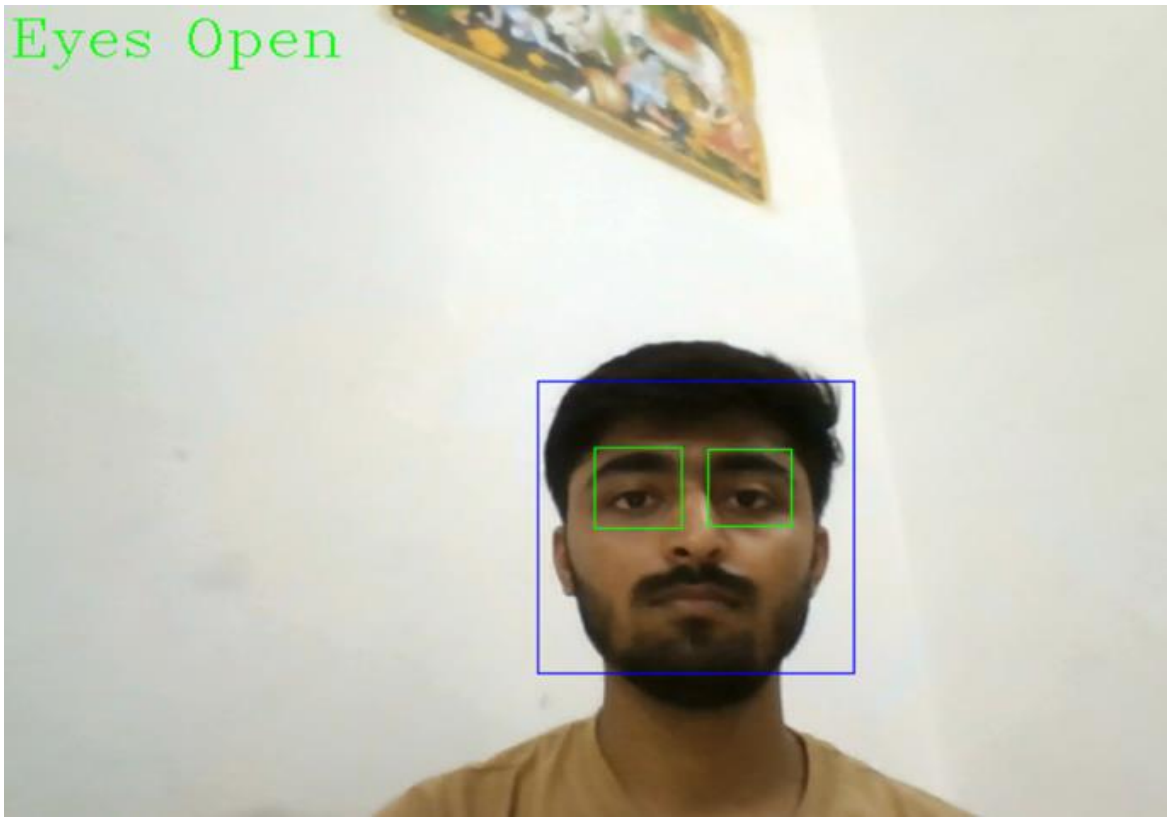

Fig 6.2 Not Drowsy(Open Eye)

Chapter 7- System Testing, Implementation & Maintenance

PERFORMANCE MEASURES

After training the model, it's performance should be measured. There are different metrics to evaluate a Classification model like Loss, Accuracy, Precision and Recall. We use Matplotlib to plot the graphs of different metrics of the model.

Some important terms in Classification Metrics are:

True Postives (TP):
It is the case where the predicted output is positive and it is true (the actual output is also postive).
True Negatives (TN):
It is the case where the predicted output is negative and it is true (the actual output is also negative).
False Postives (FP):
It is the case where the predicted output is positive but it is false (the actual output is negative).
False Negatives (FN):
It is the case where the predicted output is negative but it is false (the actual output is postive).

7.1. Loss

Loss is calculated after each epoch to measure the performance of the model. We use Categorical Cross-entropy because this is used when there are two or more label classes. This expects labels to be provided in a one-hot representation.
The graph plotted for loss on training data and validation data during the training process is as follows:

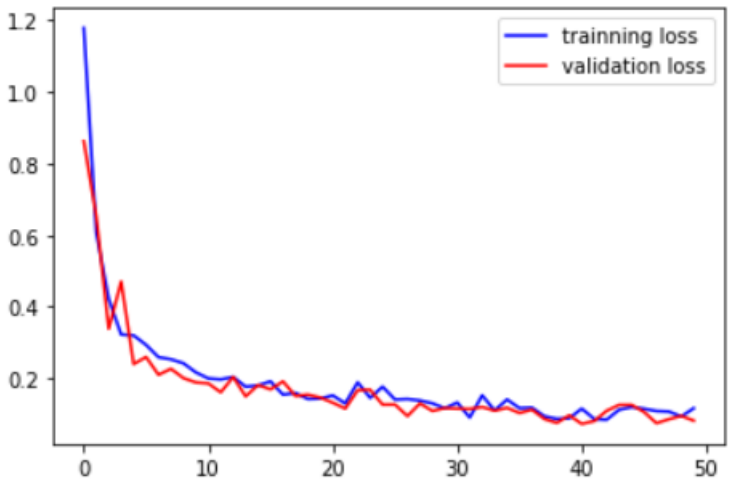

Fig 7.1 Training And Validation Loss

7.2. Accuracy Classification

accuracy is the number of correct predictions made as a ratio of all predictions made. Accuracy = No.of correct predictions / Total no.of Input samples The graph of accuarcy on training data and validation data during the training process is as follows:

Fig 7.1 Training And Validation Accuracy

Chapter 8- Summary and Future Scope

CONCLUSION AND FUTURE SCOPE

8.1. Conclusion

The paper described an drowsiness detection system based on CNN-based Machine Learning. We used OpenCV to detect faces and eyes using a haar cascade classifier and then we used a CNN model to predict the status. Finally, driver fatigue is evaluated. The experimental results demonstrate that when the drowsy counts reaches to 5, the driver can be considered in a fatigue state. The system was able to detect facial landmarks from images and pass it to a CNN-based trained Deep Learning model to detect drowsy driving behavior. There are limitations to this technology, such as obstructing the view of facial features by wearing sunglasses and bad lighting conditions. However, given the current state, there is still room for performance improvement and better facial feature detection even in bad lighting conditions. Thus we have successfully designed a prototype drowsiness detection system using OpenCV software and Haar Classifiers. The system so developed was successfully tested, its limitations identified and a future plan of action developed.

8.2. Future Scope

The work can be extended by combing physiological approach techniques, so the driver can be detected as drowsy through those readings. In special situations like when a driver wares spectacles along with mask then we can use the physiological reading of the driver. We can detect the drowsiness of the driver more accurately with these readings. We can use the devices which are mentioned earlier and detect the drowsiness of the driver. While this is a research project, there is scope when this completely turns out to be developed into an application which can be run by the end users on their own for their own purposes on their own systems.

Appendices

DROWSINESS

Drowsiness refers to feeling abnormally sleepy during the day. People who are drowsy may fall asleep in inappropriate situations or at inappropriate times.

PYTHON

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. It is created by Guido Van Rossum and first released in 1991. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

OPENCV

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by willow garage then Itseez (which was later acquired by Intel). The library is cross platform and free for use under the open source BSD license. OpenCV supports some models from deep learning like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers. It promotes Open Vision Capsules which is a portable format, compatible with all other formats.

MACHINE LEARNING

Machine Learning is defined as the study of computer programs that leverage algorithms and statistical models to learn through inference and patterns without being explicitly programed.

Training Data

The part of the data in the dataset is used to train the model. The model learn from this data. Here the training data is 80% of the dataset.

Validation Data

The part of data which is used to do a frequent evaluation of model. This data plays its part when the model is actually training. This is the 10% part of the training data.

Testing Data

Once our model is completely trained, we feed the testing data to the model for prediction. This testing data is 20% of the total dataset.

SEQUENTIAL MODEL

The Sequential model allows you to easily stack sequential layers (and even recurrent layers) of the network in order from input to output. We use the 'add()' function to add layers to our model.

SUPERVISED LEARNING

Supervised learning is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process.

MATPLOTLIB

It is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

NUMPY

It is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of highlevel mathematical functions to operate on these arrays.

PLAYSOUND

Playsound is a “pure Python, cross platform, single function module with no dependencies for playing sounds.” With this module, you can play a sound file with a single line of code: from playsound import playsound playsound (“...”).

BACK PROPAGATION

It is an algorithm commonly used to train neural networks. When the neural network is initialized, weights are set for its individual elements, called neurons. Inputs are loaded, they are passed through the network of neurons, and the network provides an output for each one, given the initial weights. Backpropagation helps to adjust the weights of the neurons so that the result comes closer and closer to the known true result.

CONVOLUTIONAL NEURAL NETWORK

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are a mathematical function that calculate the weighted sum of multiple inputs and outputs an activation value.

REGION OF INTEREST

A region of interest (ROI) is a portion of an image that you want to filter or perform some other operation on. You define an ROI by creating a binary mask, which is a binary image that is the same size as the image you want to process with pixels that define the ROI set to 1 and all other pixels set to 0.

RELU The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

References

1. Morgenthaler, T.I.; Lee-Chiong, T.; Alessi, C.; Friedman, L.; Aurora, R.N.; Boehlecke, B.; Brown, T.; Chesson, A.L., Jr.; Kapur, V.; Maganti, R.; et al. Practice parameters for the clinical evaluation and treatment of circadian rhythm sleep disorders. Sleep 2007, 30, 1445–1459. [Google Scholar] [CrossRef] [PubMed][Green Version]

2. Marjorie, V.; Heather, M.E.; Neil, J.D. Sleepiness and sleep-related accidents in commercial bus drivers. Sleep Breath. 2009, 14, 39–42. [Google Scholar]

3. Jabbar, R.; Al-Khalifa, K.; Kharbeche, M.; Alhajyaseen, W.K.M.; Jafari, M.; Jiang, S. Real-time Driver Drowsiness Detection for Android Application Using Deep Neural Networks Techniques. Procedia Comput. Sci. 2018, 130, 400–407. [Google Scholar] [CrossRef]

4. Chellappa, A.; Reddy, M.S.; Ezhilarasie, R.; Suguna, S.K.; Umamakeswari, A. Fatigue Detection Using Raspberry Pi 3. Int. J. Eng. Technol. 2018, 7, 29–32. [Google Scholar] [CrossRef][Green Version]

5. Mohana, B.; Sheela Rani, C.M. Drowsiness Detection Based on Eye Closure and Yawning Detection. Int. J. Recent Technol. Eng. 2019, 8, 1–13. [Google Scholar]

6. Wong, J.Y.; Lau, P.Y. Real-Time Driver Alert System Using Raspberry Pi. ECTI Trans. Electr. Eng. Electron. Commun. 2019, 17, 193–203. [Google Scholar] [CrossRef]

7. Ramos, A.L.A.; Erandio, J.C.; Mangilaya, D.H.T.; Carmen, N.D.; Enteria, E.M.; Enriquez, L.J. Driver Drowsiness Detection Based on Eye Movement and Yawning Using Facial Landmark Analysis. Int. J. Simul. Syst. Sci. Technol. 2019, 20, 1–8. [Google Scholar] [CrossRef]

8. Mehta, S.; Dadhich, S.; Gumber, S.; Bhatt, A.J. Real-Time Driver Drowsiness Detection System Using Eye Aspect Ratio and Eye Closure Ratio. In Proceedings of the International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Jaipur, India, 26–28 February 2019; pp. 1333–1339. [Google Scholar