CHAPTER 1

INTRODUCTION

1.1 GENERAL INTRODUCTION

According to available statistics, each year people die on the road is nearly 1.3 million and around 50 million peoples are suffering from non-fatal injuries as a result of traffic accidents. At the wheel if a driver falls asleep, the car loses its control and crashes into another vehicle or stationary objects. The driver's level of drowsiness should be monitored to stop these fatal accidents. The following tools have been widely used to track drowsiness:

1)Vehicle based measures: Deviations from lanes, steering wheel rotation, accelerator pedal pressure, and other factors are continuously tracked, and any change that exceeds a predetermined level signals indicates that the driver is drowsy.

2) Behavioural measures: A camera tracks the driver's actions, such as eye closing, eye twitch, head posture, and yawning, and if any of these drowsiness signs are observed the driver is alerted.

3)Physiological measures: Many studies have looked into the relationship between physiological signals (ECG, EMG, EOG and EEG) and driver drowsiness.

Also, another common issue with the current machine vision models is the fact that most of these algorithms are bulky (large in size) and require dedicated hardware to run the models that have been developed. They do not run efficiently on devices with low computational power.

The aim of this study is to detect and alert the person when eyes are not opened for a particular duration with the use of efficient and fast CNN model. When drowsiness is detected, this system will notify the about the drowsiness of the driver.

1.1.1 Road Accidents

Road safety is a significant public health issue, and a cause of injuries and fatalities. According to a report by the Ministry of Road Transport and Highways Transport Research Wing, road accidents claimed 1,53,972 lives and harmed 3,84,448 people in 2021. Unfortunately, the age range that is most severely hit by road accidents is 18 to 45 years old, which accounts for almost 67 percent of all accidental deaths.

A crucial problem that causes numerous car accidents annually is driver fatigue. Due to the incapacity of a driver to halt or swerve to prevent or minimise the impact, accidents caused by driver sleepiness are much more inclined to result in fatalities or severe accidents. Fatigue lowers attentiveness, alertness, and concentration, which impairs the accomplishment of tasks requiring attention such as driving.

Measures that India ought to take to prevent accidents caused by driver fatigue

- **Diagnosing sleep disorders:** Sleeping is one of the most fundamental needs for humans. The inability to consistently get enough sleep due to multiple conditions or sleep disorders significantly affects the quality of sleep for many people. Sleep problems can impair the capacity to function both during the day and at night. Obstructive sleep apnea (OSA), one of the sleep disorders, has been linked by research, to excessive daytime sleepiness (EDS). Drowsiness and daytime sleepiness while driving is a common symptom of many sleep disorders. Diagnosing sleep disorders and taking action are essential for driver and pedestrian safety. With Cloud-connected CPAP machines, devices, and digital health innovations used to manage sleep apnea, ResMed's machines help individuals breathe more comfortably and improve sleep while also attempting to transform care for patients with sleep apnea, COPD, and other chronic conditions.

- **Incorporate detectors and AI to recognize driver sleepiness:** Systems that assist in monitoring driver fatigue, use sensors to observe a driver's eye movements, breathing patterns, and yawning to detect when they are exhausted and warn them in time to prevent an accident. Frequent yawning, drowsiness that comes and goes, and missing turns are just a few indications of drowsy driving that should notify people to the need of stop driving.
- **Education:** The "Grant of Financial Assistance for Administering Road Safely Advocacy and Awards for the Outstanding Work Done in the Field of Road Safety" programme has been put into place by the Ministry of Road Transport & Highways. Financial aid is given under the curriculum to a variety of eligible organisations, which include non-profit organisations, cooperative societies, firms, and academic institutions accredited under the UGC Act. The Road Safety Audit, Pilot Projects, Awareness Campaigns (Awareness Building, Safer Vehicles, Safer Road Users), and Capacity Building are the programme themes covered by the scheme, as specified in the scheme guidelines.
- **Road Safety Audits:** All road projects must undertake a Road Safety Audit at every stage, including planning, development, administration, and upkeep. The Road Safety Audit is being conducted in accordance with the relevant guidelines established by the Indian Road Congress (IRC).
- **Refraining from using medications and drinking before driving:** The Drivers should not consume any medicines or drugs while or before driving
- **Monitoring driver hours and budgeting time for rest:** According to the advice of experts in road safety, drivers shouldn't operate a vehicle for longer than three hours without rest of 15 to 30 minutes. A driver should not spend more than eight hours behind the wheel in a single day to maintain alertness. In order to prevent accidents and driver drowsiness, organizing intervals in drive time is imperative.
- **Emergency care:** Emergency services should be supplied by ambulances, paramedics, patrol cars, helicopters, and safe driving should be enforced by traffic cameras, CCTVs, dashboard cameras, and other devices.

According to data from the World Road Statistics, 2020, mentioned in the report of the Ministry of Road Transport and Highways Transport Research Wing, India is 2nd among 207 nations in terms of the total number of accidents. The World Road Statistics, 2020 lists 207 countries, with India having the highest number of fatalities. Thus, for National Road Safety Week which is observed on 11th to 17th January, to effectively safeguard oneself and other people from drowsy driving, individuals must adhere to improving sleep health and routines. Prioritizing sleep is crucial for maintaining overall health and well-being.

1.2 TECHNOLOGIES USED

1.2.1 Machine Learning

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values.

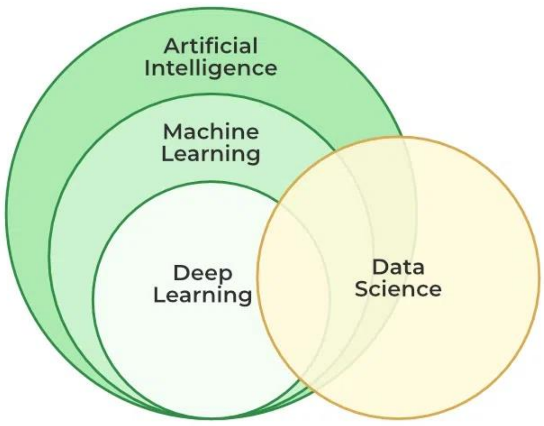


Figure 1.1 Machine Learning

Types of Machine Learning

- **Supervised learning:** In this type of machine learning, data scientists supply algorithms with labeled training data and define the variables they want the algorithm to assess for correlations. Both the input and the output of the algorithm is specified.

- **Unsupervised learning:** This type of machine learning involves algorithms that train on unlabeled data. The algorithm scans through data sets looking for any meaningful connection. The data that algorithms train on as well as the predictions or recommendations they output are predetermined.
- **Semi-supervised learning:** This approach to machine learning involves a mix of the two preceding types. Data scientists may feed an algorithm mostly labeled training data, but the model is free to explore the data on its own and develop its own understanding of the data set.
- **Reinforcement learning:** Data scientists typically use reinforcement learning to teach a machine to complete a multi-step process for which there are clearly defined rules. Data scientists program an algorithm to complete a task and give it positive or negative cues as it works out how to complete a task. But for the most part, the algorithm decides on its own what steps to take along the way.

Seven steps of Machine Learning

1. Gathering Data
2. Preparing that data
3. Choosing a model
4. Training
5. Evaluation
6. Hyperparameter Tuning
7. Prediction

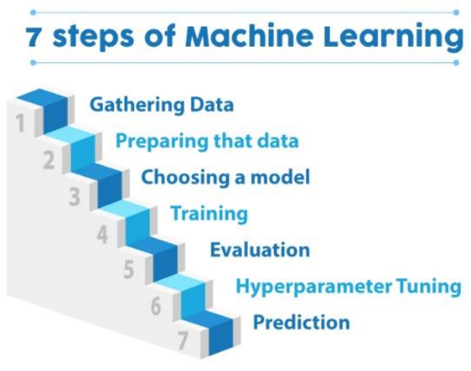


Figure 1.2 Steps in Machine Learning

How does Machine Learning work?

The three major building blocks of a system are the model, the parameters, and the learner.

- Model is the system which makes predictions
- The parameters are the factors which are considered by the model to make predictions
- The learner makes the adjustments in the parameters and the model to align the predictions with the actual results

1. Learning from the training set

This involves taking a sample data set of several drinks for which the colour and alcohol percentage is specified. Now, description of each classification is defined, that is wine and beer, in terms of the value of parameters for each type. The model can use the description to decide if a new drink is a wine or beer.

The values of the parameters, 'colour' and 'alcohol percentages' is represented as 'x' and 'y' respectively. Then (x,y) defines the parameters of each drink in the training data. This set of data is called a training set. These values, when plotted on a graph, present a hypothesis in the form of a line, a rectangle, or a polynomial that fits best to the desired results.

2. Measure error

Once the model is trained on a defined training set, it needs to be checked for discrepancies and errors. A fresh set of data to accomplish this task is used. The outcome of this test would be one of these four:

- True Positive: When the model predicts the condition when it is present
- True Negative: When the model does not predict a condition when it is absent

- False Positive: When the model predicts a condition when it is absent
- False Negative: When the model does not predict a condition when it is present

  The sum of FP and FN is the total error in the model.

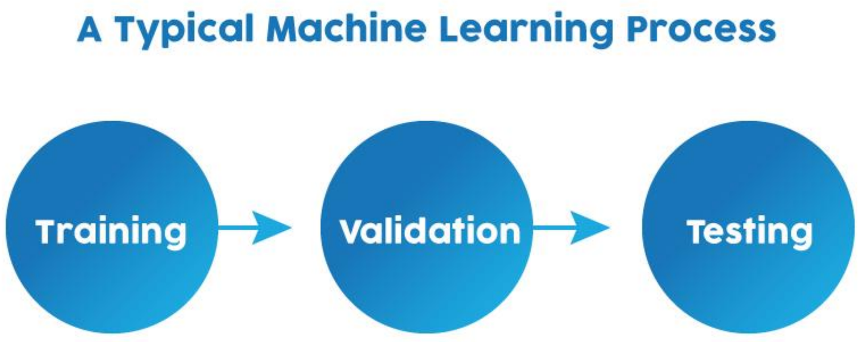


Figure 1.3 Machine Learning Process

3. Manage Noise

For the sake of simplicity, only two parameters is considered to approach a machine learning problem here that is the colour and alcohol percentage. But in reality, hundreds of parameters and a broad set of learning data will be considered to solve a machine learning problem.

The hypothesis then created will have a lot more errors because of the noise. Noise is the unwanted anomalies that disguise the underlying relationship in the data set and weakens the learning process. Various reasons for this noise to occur are:

- Large training data set
- Errors in input data
- Data labelling errors
- Unobservable attributes that might affect the classification but are not considered in the training set due to lack of data.

4. Testing and Generalization

While it is possible for an algorithm or hypothesis to fit well to a training set, it might fail when applied to another set of data outside of the training set. Therefore, it is essential to figure out if the algorithm is fit for new data. Testing it with a set of new data is the way to judge this. Also, generalisation refers to how well the model predicts outcomes for a new set of data.

When hypothesis algorithm is fit for maximum possible simplicity, it might have less error for the training data, but might have more significant error while processing new data. This is called underfitting. On the other hand, if the hypothesis is too complicated to accommodate the best fit to the training result, it might not generalise well. This is the case of over-fitting. In either case, the results are fed back to train the model further.

1.2.2 Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It’s achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers.

How does deep learning attain such impressive results?

Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safety-critical applications like driverless cars. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images.

While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

- Deep learning requires large amounts of **labelled data**. For example, driverless car development requires millions of images and thousands of hours of video.
- Deep learning requires substantial **computing power**. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

How Deep Learning Works

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

The term "deep" usually refers to the number of hidden layers in the neural network. Traditional neural networks (4:37) only contain 2-3 hidden layers, while deep networks can have as many as 150.

Deep learning models are trained by using large sets of labelled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

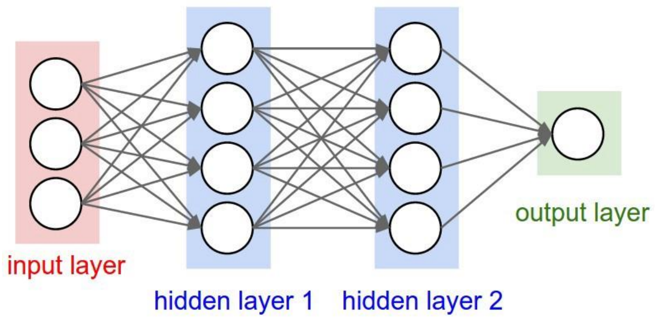


Figure 1.4 Deep Learning Layers

Steps of Deep Learning Process

- **Acquiring data**

  The ability to acquire data can make or break the solution. Not only is getting data usually the most important part of a deep learning project, but it’s also often the hardest.

- **Preprocessing**

  At a high-level,when preprocessing data for neural networks, the following is done : 1) clean the data, 2) handle categorical features and text and 3) scale the real-valued features using normalization or standardization techniques.

- **Splitting and balancing the dataset**

  Generally, the data is split into two datasets: training and validation. In some cases, a third holdout dataset is created, referred to as the test set.

- **Building and training the model**

  For each layer, a reasonable number of hidden units is selected. There is no absolute science to choosing the right size for each layer, nor the number of layers -it all depends on your specific data and architecture.

- **Evaluation**

  Each time the model is trained, its performance is evaluated on the validation set. When a validation set at training time is provided, Keras handles this automatically. The performance on the validation set gives us a sense for how the model will perform on new, unseen data.

- **Deploying our solution**

  Once the model is trained, it is deployed into the real world. This is especially true in industry settings, when the networks will be used by co-workers and customers, or working behind the scenes in our products and internal tools.

1.2.3 Convolutional Neural Networks (CNN)

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

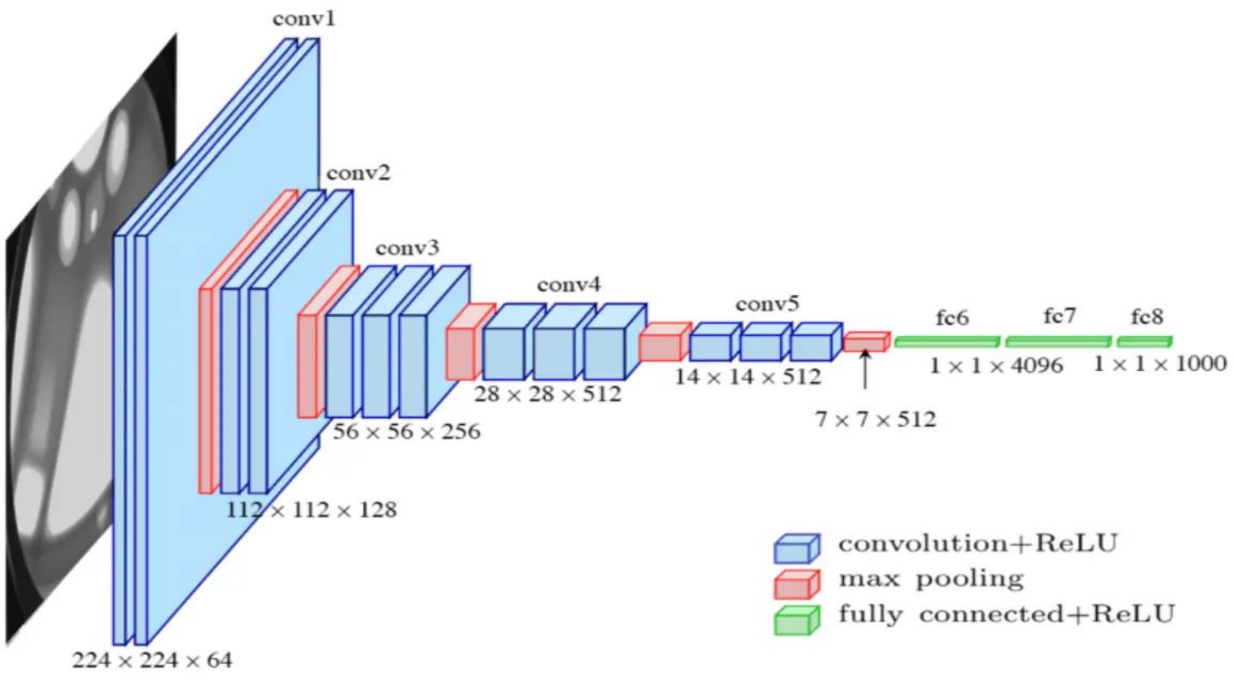


Figure 1.5 CNN Architecture

A CNN is composed of an input layer, an output layer, and many hidden layers in between as in Figure 1.5.

These layers perform operations that alter the data with the intent of learning features specific to the data. Three of the most common layers are convolution, activation or ReLU, and pooling.

- **Convolution** puts the input images through a set of convolutional filters, each of which activates certain features from the images.
- **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as *activation*, because only the activated features are carried forward into the next layer.

- **Pooling** simplifies the output by performing nonlinear down sampling, reducing the number of parameters that the network needs to learn.

1.2.3.1 Example of CNN:

Consider the image in Figure 1.6.

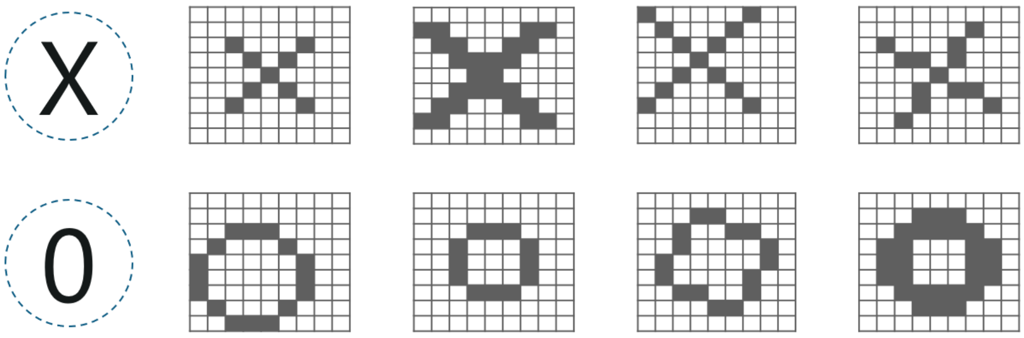
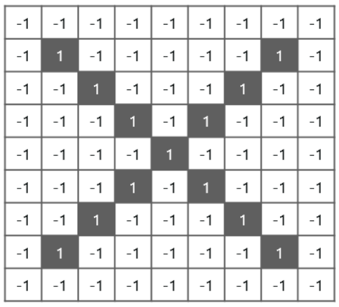


Figure 1.6 Example of CNN

Here, there are multiple renditions of X and O's. This makes it tricky for the computer to recognize. But the goal is that if the input signal looks like previous images it has seen before, the "image" reference signal will be mixed into, or convolved with, the input signal. The resulting output signal is then passed on to the next layer as in Figure 1.7.



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 1.7 Pixels in CNN

So, the computer understands every pixel. In this case, the white pixels are said to be -1 while the black ones are 1. A basic binary classification to differentiate the pixels is implemented.

1.2.3.2 Convolution Of An Image

Convolution has the nice property of being translational invariant. Intuitively, this means that each convolution filter represents a feature of interest (e.g pixels in letters) and the Convolutional Neural Network algorithm learns which features comprise the resulting reference (i.e. alphabet).

There are 4 steps for convolution:

- Line up the feature and the image
- Multiply each image pixel by corresponding feature pixel
- Add the values and find the sum
- Divide the sum by the total number of pixels in the feature

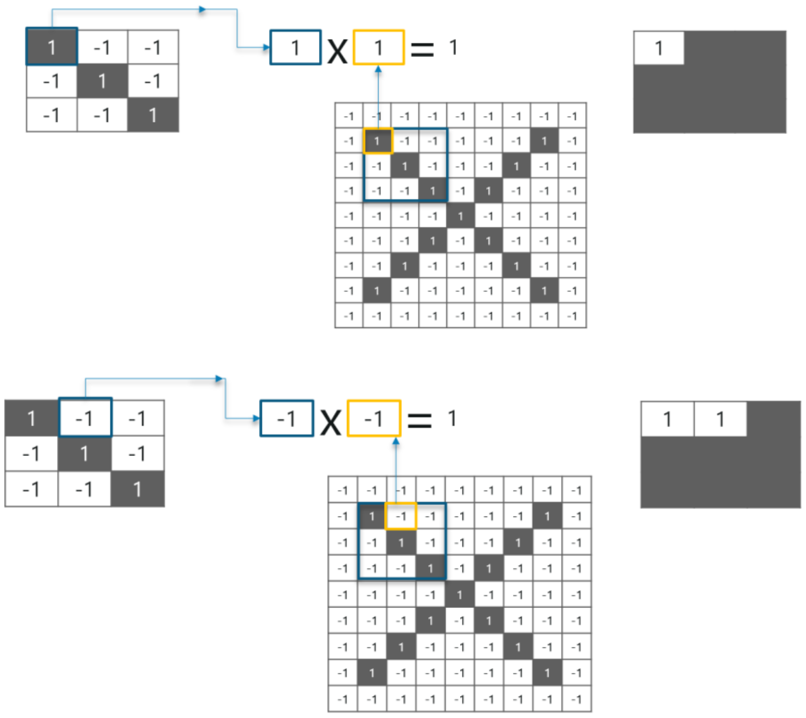
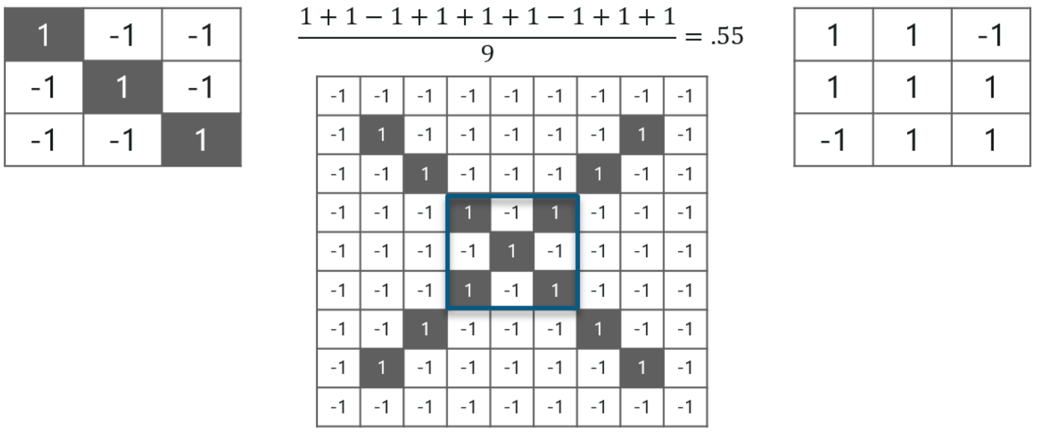


Figure 1.8 Convolution of Image

Consider the above Figure 1.8. A feature image is considered and one pixel from it. It is multiplied with the existing image and the product is stored in another buffer feature image. The values are added which led to the sum. Then, divide this number by the total number of pixels in the feature image. When that is done, the final value obtained is placed at the center of the filtered image as shown below:

Now, this filter can be moved around and do the same at any pixel in the image. For better clarity, see figure 1.9.



1	-1	-1
-1	1	-1
-1	-1	1

$$\frac{1+1-1+1+1+1-1+1+1}{9} = .55$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	-1
1	1	1
-1	1	1

Figure 1.9 CNN Algorithm

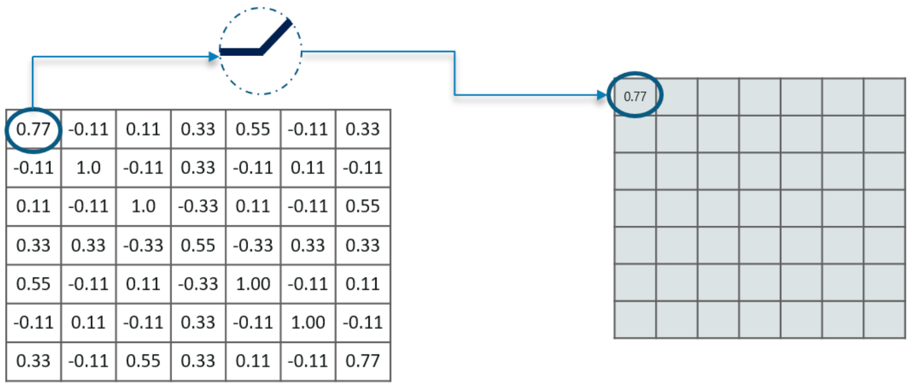
The output signal strength is not dependent on where the features are located, but simply whether the features are present. Hence, an alphabet could be sitting in different positions and the **Convolutional Neural Network algorithm** would still be able to recognize it.

1.2.3.3 ReLU Layer

ReLU is an activation function. But, what is an activation function?

Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

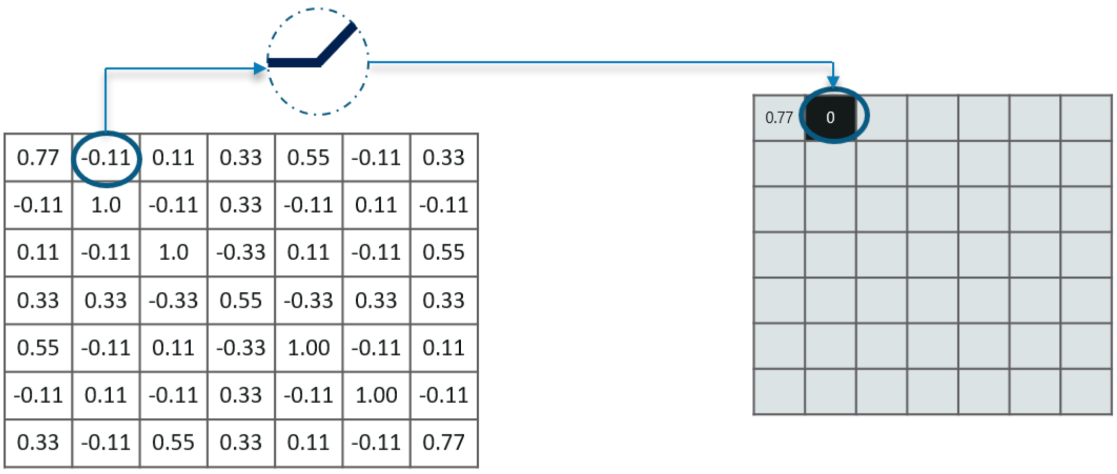
Why is ReLU required? (See figure 1.10)



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Figure 1.10 ReLU

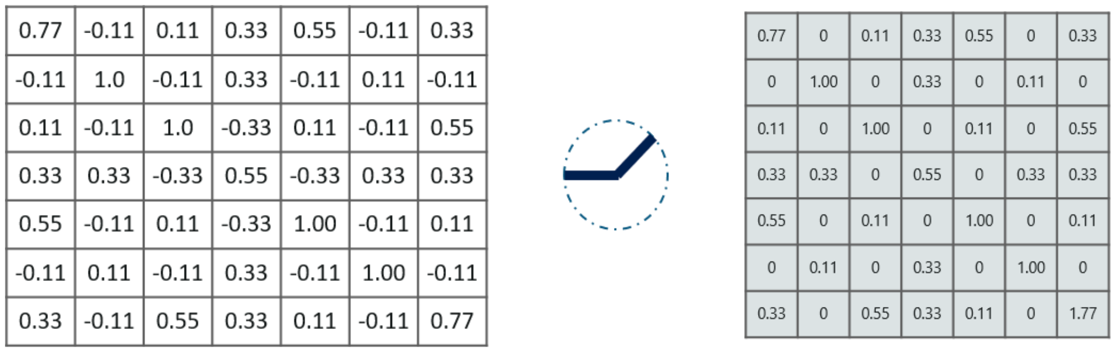
The main aim is to remove all the negative values from the convolution. All the positive values remain the same but all the negative values get changed to zero as shown below in figure 1.11:



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Figure 1.11 Function of Activation Layer

So after this particular feature is processed, the following output is found in figure 1.12:



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

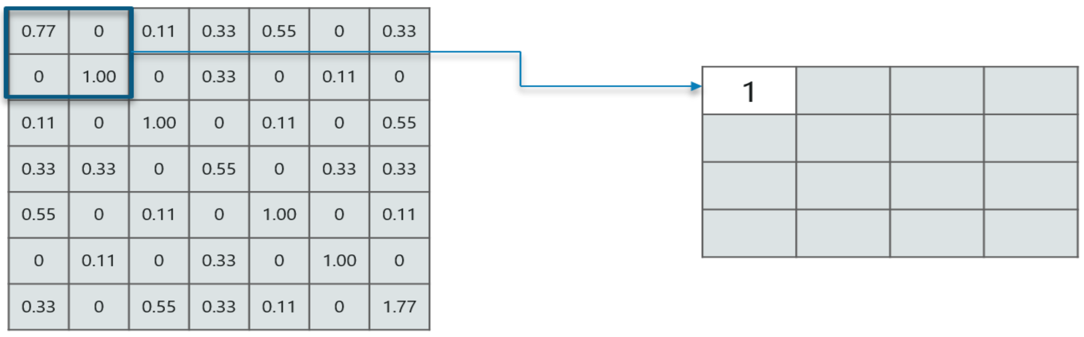
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

Figure 1.12 Output of ReLU layer

1.2.3.4 Pooling Layer

In this layer, shrink the image stack into a smaller size. Pooling is done after passing through the activation layer as in Figure 1.13.

- Pick a window size (usually 2 or 3)
- Pick a stride (usually 2)
- Walk your window across your filtered images
- From each window, take the maximum value



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

1			

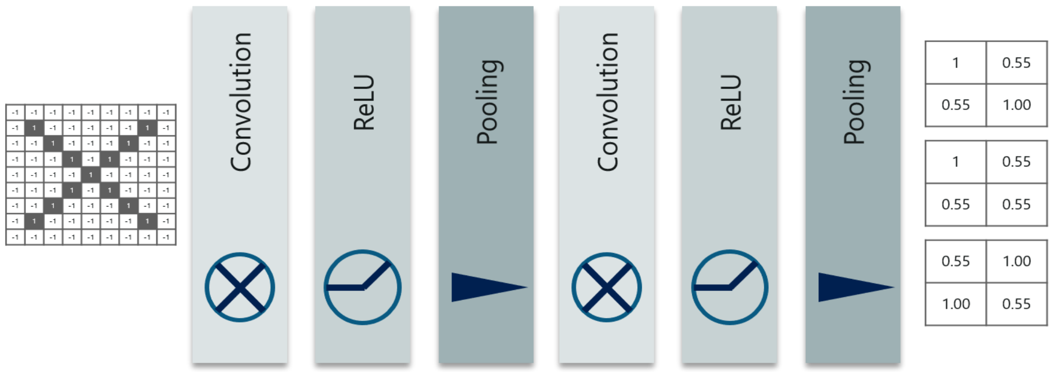
Figure 1.13 Pooling layer

So in this case, window size is 2 and 4 values to choose from. From those 4 values, the maximum value there is 1, pick 1. Also, note that it started out with a 7×7 matrix but now the same matrix after pooling came down to 4×4.

But the window across the entire image is moved. The procedure is exactly as same as above and the process for entire image is repeated.

1.2.3.5 Stacking Up The Layers

So to get the time-frame in one picture, a 4×4 matrix from a 7×7 matrix after passing the input through 3 layers – Convolution, ReLU and Pooling as shown in Figure 1.14.



1	0.55
0.55	1.00

1	0.55
0.55	0.55

0.55	1.00
1.00	0.55

Figure 1.14 Stacking up of layers

The last layers in the network are fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers. Also, fully connected layer is the final layer where the classification actually happens.

1.2.3.6 Dropout in Neural Networks

The term “**dropout**” refers to dropping out units (both hidden and visible) in a neural network.

Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. These units are not considered during a particular forward or backward pass.

At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

Why is Dropout needed?

Dropout is needed “to prevent over-fitting”. A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

Training Phase

Training Phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).

Testing Phase

Use all activations, but reduce them by a factor *p* as in Figure 1.15.

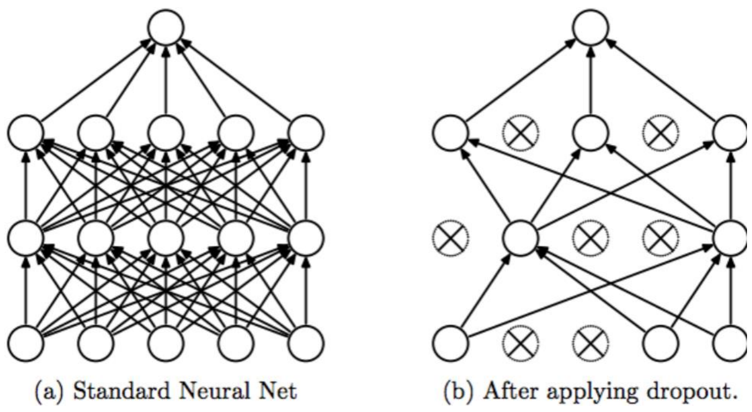


Figure 1.15 Testing Phase

Some Observations

1. Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
2. Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
3. With H hidden units, each of which can be dropped, 2^H possible models is created . In testing phase, the entire network is considered and each activation is reduced by a factor *p*.

Flatten Layers

Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4)

Dense layer

The **dense layer** is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models. In the background, the dense layer performs a matrix-vector multiplication.

Applications

- **Medical Imaging:** CNNs can examine thousands of pathology reports to visually detect the presence or absence of cancer cells in images.
- **Audio Processing:** Keyword detection can be used in any device with a microphone to detect when a certain word or phrase is spoken (“Hey Siri!”). CNNs can accurately learn and detect the keyword while ignoring all other phrases regardless of the environment.

- **Object Detection:** Automated driving relies on CNNs to accurately detect the presence of a sign or other object and make decisions based on the output.
- **Synthetic Data Generation:** Using Generative Adversarial Networks (GANs), new images can be produced for use in deep learning applications including face recognition and automated driving.

1.2.4 Computer Vision

Computer vision is one of the fields of **artificial intelligence** (leverages computers and machines to mimic the problem-solving and decision-making capabilities of the human mind) that trains and enables computers to understand the visual world. Computers can use digital images and deep learning models to accurately identify and classify objects and react to them.

Computer vision in AI is dedicated to the development of automated systems that can interpret visual data (such as photographs or motion pictures) in the same manner as people do. The idea behind computer vision is to instruct computers to interpret and comprehend images on a pixel-by-pixel basis. This is the foundation of the computer vision field. Regarding the technical side of things, computers will seek to extract visual data, manage it, and analyse the outcomes using sophisticated software programs.

1.2.4.1 How Does Computer Vision Work?

Massive amounts of information are required for computer vision. Repeated data analyses are performed until the system can differentiate between objects and identify visuals. Deep learning, a specific kind of machine learning, and convolutional neural networks, an important form of a neural network, are the two key techniques that are used to achieve this goal.

With the help of pre-programmed algorithmic frameworks, a machine learning system may automatically learn about the interpretation of visual data. The model can learn to distinguish between similar pictures if it is given a large enough dataset. Algorithms make it possible for the system to learn on its own, so that it may replace human labour in tasks like image recognition.

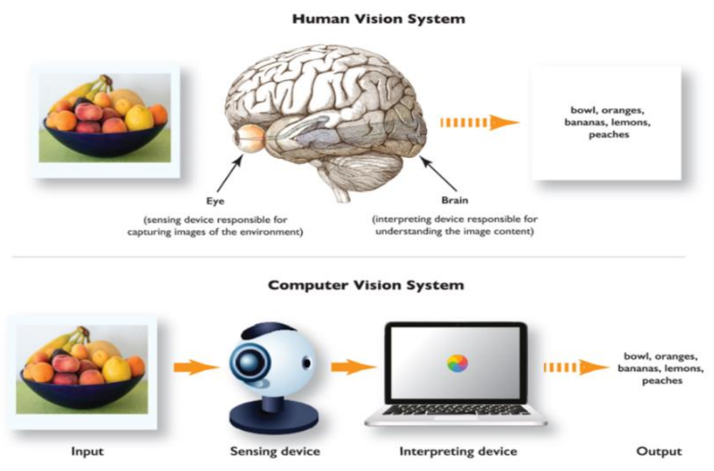

Human Vision System
bowl, oranges, bananas, lemons, peaches
Eye
(sensing device responsible for capturing images of the environment)
Brain
(interpreting device responsible for understanding the image content)
Computer Vision System
bowl, oranges, bananas, lemons, peaches
Input
Sensing device
Interpreting device
Output

Figure 1.16 Human Vision VS Computer Vision

Convolutional neural networks aid machine learning and deep learning models in understanding by dividing visuals into smaller sections that may be tagged. With the help of the tags, it performs convolutions and then leverages the tertiary function to make recommendations about the scene it is observing. With each cycle, the neural network performs convolutions and evaluates the veracity of its recommendations. And that's when it starts perceiving and identifying pictures like a human. Through a series of filtering and actions, computers can put all the parts of the image together and then think on their own.

1.2.4.2 Computer Vision Applications

One field of Machine Learning where fundamental ideas are already included in mainstream products is computer vision. The applications include:

- **Self-Driving Cars**

With the use of computer vision, autonomous vehicles can understand their environment. Multiple cameras record the environment surrounding the vehicle, which is then sent into computer vision algorithms that analyses the photos in perfect sync to locate road edges, decipher signposts, and see other vehicles, obstacles, and people. Then, the autonomous vehicle can navigate streets and highways on its own, swerve around obstructions, and get its passengers where they need to go safely.

- **Facial Recognition**

  Facial recognition programs, which use computer vision to recognize individuals in photographs, rely heavily on this field of study. Facial traits in photos are identified by computer vision algorithms, which then match those aspects to stored face profiles. In order to verify the identity of the people using consumer electronics, face recognition is increasingly being used. Facial recognition is used in social networking applications for both user detection and user tagging. For the same reason, law enforcement uses face recognition software to track down criminals using surveillance footage.

- **Augmented & Mixed Reality**

  Augmented reality, which allows computers like smartphones and wearable technology to superimpose or embed digital content onto real-world environments, also relies heavily on computer vision. Virtual items may be placed in the actual environment through computer vision in augmented reality equipment. In order to properly generate depth and proportions and position virtual items in the real environment, augmented reality apps rely on computer vision techniques to recognize surfaces like table tops, ceilings, and floors.

1.3 OBJECTIVES OF THE PROJECT

- The Objective of this project is to develop a system that can accurately detect the signs of drowsiness or fatigue in a driver such as changes in eye movement, facial expressions and head position.
- Integrating this system into a vehicle in a way that is unobtrusive and does not distract the driver.
- Providing the alerts to the driver when drowsiness is detected, such as through an audible alarm or visual warning on the dashboard.

1.4 SCOPE OF THE PROJECT

Scope of this project can vary depending on the specific goals of the project and the resources available.

- Researching and selecting appropriate sensors and technology to detect drowsiness and fatigue, such as eye-tracking devices, facial recognition software, and heart rate monitors.
- Designing and developing an algorithm to analyse the data collected by the sensors and accurately identify signs of drowsiness.
- Creating a prototype of the drowsiness detection system and testing it in a controlled environment to ensure that it is accurate and effective.
- Integrating the system into a vehicle, such as by developing a hardware device that can be installed in the car or by leveraging existing technology like cameras and sensors that are already present in many modern vehicles..

The scope of the project may also involve considering factors such as cost, scalability, and regulatory compliance, depending on the intended use case and market for the drowsiness detection system.

1.5 ORGANIZATION OF THE PROJECT

The rest of the report is organized as follows :

Chapter 1 presents the general introduction of the module.

Chapter 2 presents the related contemporary works done in the area and discusses their advantages and limitations.

Chapter 3 presents the existing system and works done in proposed system.

Chapter 4 presents the system architecture and description of the working model.

Chapter 5 presents the software description used in the system.

Chapter 6 presents the results obtained in the system.

Chapter 7 presents the conclusion and the future works of the system.

CHAPTER 2

LITERATURE SURVEY

2.1 SCOPE OF SURVEY

The following survey papers are taken for Driver Drowsiness Detection Techniques. The most popular of the existing techniques has been discussed as follows.

2.2 LITERATURE REVIEW

2.2.1 A Comparison of Fast, Surf, Eigen, Harris, and Mser features

In this work, Farman Ali, Sajid Ullah Khan, Muhammad Zarrar Mahmudi, and Rahmat Ullah compares five popular feature detection and extraction methods - FAST, SURF, Eigen, Harris, and MSER - for their effectiveness in detecting and describing features in images. The authors evaluate the performance of the methods on several datasets using various metrics, including detection rate, false-positive rate, and repeatability rate.

2.2.2 A Driver drowsiness warning system using visual information for both diurnal and nocturnal illumination conditions

In this work, M. Flores, J. Armingol, and A. de la Escalera, proposes a driver drowsiness warning system that can work under both diurnal and nocturnal illumination conditions. The system uses visual information from a camera mounted on the dashboard to detect driver drowsiness based on the frequency and duration of eyelid closures. The authors evaluate the proposed system using a dataset of 10 subjects and report a detection accuracy of 83% for daytime conditions and 80% for nighttime conditions. The authors also compare the performance of their proposed system with several existing methods and show that their system outperforms the others in terms of accuracy, sensitivity, and specificity. The proposed system can provide an effective and low-cost solution for preventing accidents caused by drowsy driving.

2.2.3 A new real-time eye tracking based on nonlinear unscented Kalman filter for monitoring driver fatigue

In this work, Z. Zhang and J. Zhang proposes a new real-time eye tracking system based on a nonlinear unscented Kalman filter for monitoring driver fatigue. The system uses a camera mounted on the dashboard to track the driver's eyes and estimate their gaze direction. The authors evaluate the proposed system using a dataset of 6 subjects and report an accuracy of 94.7% for gaze direction estimation. The authors also compare the performance of their proposed system with several existing methods and show that their system outperforms the others in terms of accuracy and robustness. The proposed system can provide an effective and reliable solution for detecting driver fatigue and preventing accidents caused by drowsy driving.

2.2.4 Comparative Analysis of Vehicle-Based and Driver-Based Features for Driver Drowsiness Monitoring by Support Vector Machines [4]

In this work, Mohamed Hedi Baccour, Franke Driewer, Tim Schack and Enkelejda Kasneci provided valuable insights into the field of driver drowsiness monitoring by conducting a comparative analysis between an indirect, direct and hybrid DMS. The results have shown that the direct DMS significantly outperforms the indirect DMS, which highlights the contributions of the driver monitoring camera and the importance of tracking eyelid and head movements to detect driver drowsiness. The best results were achieved by the hybrid DMS, which emphasizes the potential of sensor fusion to monitor driver drowsiness. Using hybrid DMSs will also help to overcome the limitations of each sensor type in different situations, e.g., unfavourable light, road or weather conditions, and make the system more robust. As they only used in that work non-intrusive measures that can be computed by means of unobtrusive in-vehicle measurement sensors, our work can help further the development of more reliable and robust driver drowsiness monitoring systems.

2.2.5 Drowsy driver detection through facial movement analysis [3]

In this work, M. Lew, N. Sebe, T. Huang, E. Bakker, E. Vural, M. Cetin, A. Ercil, G. Littlewort, M. Bartlett, and J. Movellan presents a method for detecting drowsiness in drivers based on facial movement analysis. The authors analyze facial movements of subjects in a driving simulator using facial action coding system (FACS) and compare them with baseline measurements to detect signs of drowsiness. The authors evaluate the proposed method using several performance metrics, including accuracy, sensitivity, and specificity, and report high accuracy rates in detecting drowsiness. The authors evaluate the proposed method using several performance metrics, including accuracy, sensitivity, and specificity, indicating that the method is effective for real-time drowsiness detection in driving applications.

2.2.6 Face detection and recognition with surf for human-robot Interaction [7]

In this work, Shan An, Xin Ma, Rui Song, and Yibin Li proposed a three-stage framework for face detection, feature extraction, and recognition, which integrates Speeded-Up Robust Features (SURF) algorithm with a support vector machine (SVM) classifier. The experiments conducted by the authors demonstrate the effectiveness of the proposed method in terms of high accuracy and efficiency in detecting and recognizing faces, making it a useful tool for human-robot interaction applications. However, the paper does not provide specific accuracy results for the face detection and recognition method using the SURF algorithm, and the accuracy of the method may vary depending on various factors such as the quality of the input images, the training dataset, and the specific implementation of the algorithm.

2.2.7 Facial feature detection using Haar classifiers [5]

In this work, Phillip Ian Wilson and Dr John Fernandez presents a method for detecting facial features such as eyes, nose, and mouth using Haar classifiers and the Viola-Jones algorithm. The authors report high accuracy results, with an average of 95% for eye detection, 90% for nose detection, and 85% for mouth detection based on experiments conducted on a dataset of 100 images. However, the accuracy of the method may vary depending on various factors such as the quality of the input images, the training dataset, and the specific implementation of the algorithm. Overall, the proposed method is a useful tool for applications such as face recognition and tracking.

2.2.8 Face Recognition Based on Facial Features

In this work, Muhammad Sharif, Muhammad Younus Javed, and Sajjad Mohsin presents a method for face recognition based on facial features, using Gabor wavelet transform and principal component analysis (PCA) for feature extraction and classification, respectively. The authors report high accuracy in face recognition based on the experiments conducted using a dataset of 400 images, making it a useful tool for access control and surveillance applications. However, the specific accuracy result for the proposed method is not provided, and the accuracy of the method may vary depending on various factors such as the quality of the input images, the size of the dataset, and the specific implementation of the algorithm.

2.2.9 Multiscale dynamic features based driver fatigue detection [9]

In this work, Yin, B.-C.; Fan, X.; Sun, Y.-F proposes a method for detecting driver fatigue using multiscale dynamic features. The proposed method analyses the dynamic features of facial images captured by a camera mounted on the dashboard of a car to detect the signs of driver fatigue. The authors evaluate the proposed method using a dataset of 15 subjects and report an average accuracy of 84.13% in detecting driver fatigue. The authors also compare the performance of their proposed method with several existing methods and show that their proposed method outperforms the others. The proposed method shows promise in real-world applications for detecting driver fatigue and preventing accidents caused by drowsy driving.

2.2.10 Yawning detection based on Gabor wavelets and LDA [8]

In this work, F. Xiao, C.Y. Bao, and F.S. Yan proposes a method for yawning detection based on Gabor wavelets and linear discriminant analysis (LDA). The authors describe the underlying principles of Gabor wavelets and LDA and use these techniques to extract relevant features from facial images and classify them as yawning or non-yawning. The experiments conducted by the authors show that the proposed method achieves a high accuracy rate in yawning detection, making it a promising tool for drowsiness detection and other related applications.

2.3 SUMMARY

Driver drowsiness detection systems have been extensively researched, utilizing physiological and behavioural approaches combined with sensor technologies and machine learning algorithms. These systems have the potential to significantly reduce accidents caused by driver fatigue. Future research could focus on improving accuracy, robustness, and real-time performance of detection systems. Additionally, integrating these systems with advanced vehicle technologies would further enhance road safety and driver well-being.

CHAPTER 3

SYSTEM ANALYSIS

3.1 INTRODUCTION

The primary goal of system analysis is to gain a comprehensive understanding of a system, including its objectives, requirements, constraints, and stakeholders. This understanding is achieved through a combination of observation, data collection, modeling, and evaluation techniques. By breaking down a system into its constituent parts and studying their relationships, system analysts can identify areas for improvement and inefficiencies.

3.2 ESTIMATION OF DRIVER VIGILANCE STATUS USING REAL-TIME FACIAL EXPRESSION AND DEEP LEARNING

The existing system is composed of two sequential subsystems; the input subsystem performs face detection and preprocessing of real-time input video stream data utilizing Haar cascade algorithm, and the output performs feature extraction and image classification using CNN **LeNet** architecture. The evaluation of the model using stratified five-fold cross validation on UTA-RLDD showed that the model achieved high values of average accuracy, precision, recall, and F1-score, which are 0.918, 0.928, 0.920, and 0.920, respectively. The system employs deep learning algorithms to analyze the real-time facial expressions of the driver captured through a camera installed in the vehicle. By detecting and interpreting facial cues, such as eye closure, facial muscle movements, the system can determine the driver's level of alertness or drowsiness as in Figure 3.1.

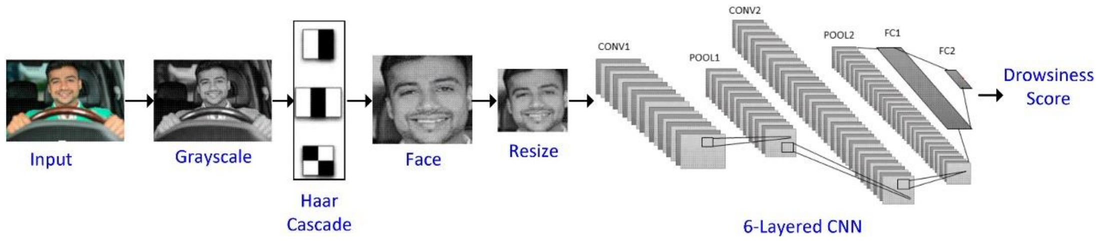


Figure 3.1 Existing Model of CNN

3.3 PROPOSED SYSTEM

Driver drowsiness detection systems are particularly important for long-haul truckers, shift workers, and other individuals who may be at higher risk of drowsy driving. These systems can help prevent accidents and improve road safety by alerting drivers to take breaks and rest when needed. In addition to preventing accidents, driver drowsiness detection systems can also improve productivity and reduce costs for fleet operators.

In this system, a camera is installed in the vehicle to track the driver's eye movements continuously. The camera captures images of the driver's eyes and uses image processing algorithms to detect changes in eye closure distance and frequency. If the system detects that the distance between the driver's eyelids is smaller than a certain threshold for a continuous period of 4 to 5 seconds, it will trigger an alert. This alert can take the form of a beep sound to alert the driver and draw their attention to the fact that they may be drowsy. In addition to the beep sound, the system can also take further action to ensure the driver's safety. Overall, this eye tracking system for driver drowsiness detection has the potential to improve driver safety and reduce the risk of accidents caused by drowsy driving. By monitoring the driver's eye movements continuously and taking appropriate action when drowsiness is detected, this system can help keep drivers safe and prevent accidents on the road.

3.4 ADVANTAGES OF PROPOSED SYSTEM

- **Improved Safety:** The primary advantage of this system is improved safety for the driver and other road users. Drowsy driving is a significant risk factor for accidents on the road, and this system can help prevent accidents by detecting drowsiness in the driver and alerting them to take a break or stop driving.
- **Real-time Monitoring:** The system continuously monitors the driver's eye closure distance in real-time, allowing for prompt detection of drowsiness. This real-time monitoring ensures that drivers are alerted to any signs of

drowsiness as soon as they occur, improving the likelihood of taking preventative action before an accident occurs.

- **Non-intrusive:** The system is non-intrusive and does not require any physical attachment to the driver's body. This means that it is comfortable to use and does not interfere with the driver's ability to operate the vehicle.
- **Automated Alerts:** The system is designed to provide automated alerts to the driver when signs of drowsiness are detected. These alerts can take the form of an audible beep or a visual warning, which can help to wake the driver up and draw their attention to the fact that they may be feeling drowsy.

3.5 SUMMARY

This chapter concludes about the existing method from which our system is improved and the method proposed to overcome the limitations and improving the accuracy from the existing system.

CHAPTER 4

SYSTEM ARCHITECTURE

4.1 INTRODUCTION

It serves as an introductory overview of a specific project, providing essential details about its purpose, objectives, scope, and deliverables. It outlines the key aspects of the project and provides a foundation for stakeholders to understand its significance and potential outcomes. The system architecture sets the stage for further planning, execution, and successful completion of the project.

4.2 SYSTEM ARCHITECTURE OF VIOLA-JONES ALGORITHM

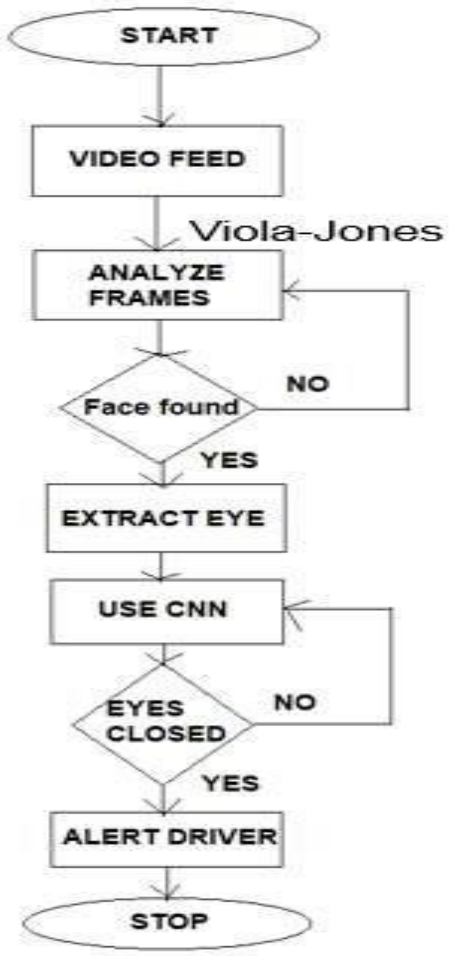


Figure 4.1 Viola-jones Algorithm Flow Chart

The input videos are fed into the code so that the CNN gets trained and validated. After training is complete, live video feed is used to analyze the eyes and mouth frame-points by Viola-Jones algorithm. Then by using CNN the colour images are converted into gray scale and due to training, the points are detected accurately. Then finally, the driver is alerted by suitable alert messages as shown in figure 4.1.

The Viola–Jones object detection framework is an algorithm which was proposed in 2001 by Paul Viola and Michael Jones. Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection.

Object detection does not need to be implemented to each frame in a video of moving objects. Instead, tracking algorithms can be used to detect and track salient features within the detection bounding boxes as they move between frames. Not only does this increase tracking speed by removing the need to re-detect features in each frame, but it also strengthens robustness because the salient features are more immune to rotation and photometric changes than the Viola-Jones detection system.

4.3 FACIAL LANDMARKS - 68

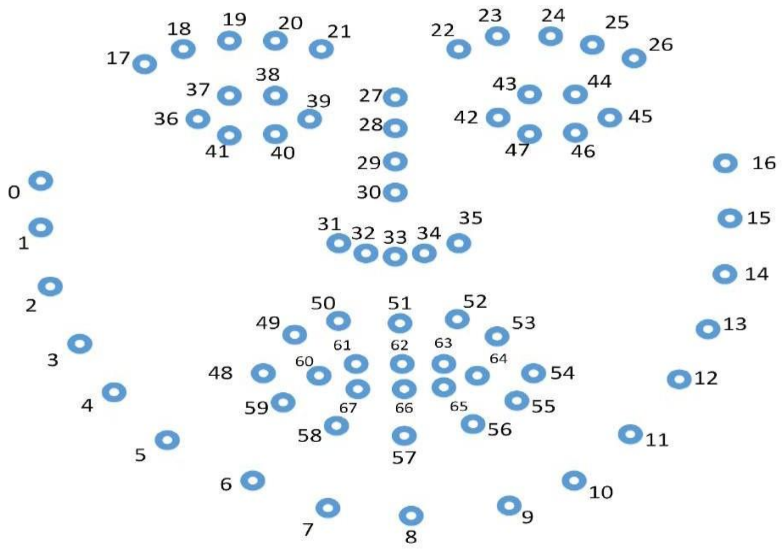


Figure 4.2 Facial Points

As shown in Figure 4.2, it depicts the 68 facial landmark points of humans. The three points of interest are

- Right eye(36-41)
- Left eye(42-47)
- Mouth(60-67)

These points are to be fed into the code so that they are detected in prior whenever necessary. Whenever the eyes are closed for some threshold period, the alarm message would pop-up in the multimedia screen.

4.4 WORKING OF VIOLA JONES ALGORITHM

The way that the Viola-Jones algorithm actually works is through the execution of four main steps:

- Haar-like Features
- Integral Image
- Adaboost Training
- Attentional cascade (Viola & Jones, 2001)

The first step, Haar-like features, are a set of rectangular digital image features that break up the image into multiple sets of “two adjacent rectangles located at any scale and position within an image” (Ephraim, Himmelman, & Siddiqi, 2009). These rectangles are then applied to the image that has been opened within the program.

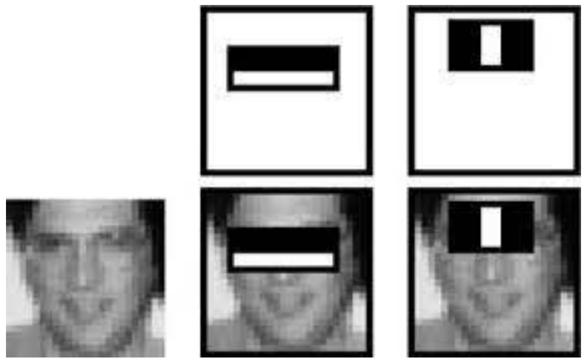


Figure 4.3 Haar Like Features

In Figure 4.3 the Haar-like features are applied to the image. As demonstrated, there is first one main region that is being examined, the area from the forehead to the eyes. This region is selected given that the region of the eyes is darker than the region of the nose and cheeks so if the Haar-like features can be matched then the image can go to the next step with the Adaboost training. What Adaboost training does here is to better define the region from the eyes to the cheeks. The way it does this is by employing a learning algorithm that is used to "teach" the program to look over a set of possible areas and then choose the areas with the "best" features resulting in a reduced image with more defined regions (Viola & Jones, 2001).

After the features are selected they are put through the Adaboost learning algorithm to narrow down the number of features that are looked at and then passed on to the cascading stage. In more broad terms, the Adaboost training makes sure that the region that is being examined is as precise as possible in order to help obtain the best accuracy. The last step is the attentional cascade. During this step, the program is trying, again, to maintain the smallest error percentage rate as possible. In order to do this, the attentional cascade's main focus is to eliminate as many false positives as possible. In practice, for instance, the initial steps identify an object, such as a bush, as a potential face then the attentional cascade takes the bush and discards it as a viable option.

The Viola-Jones method, although it has a high accuracy rate, does come at the price of longer computation time. In contrast, CNNs have addressed computation time concerns and have successfully improved upon them.

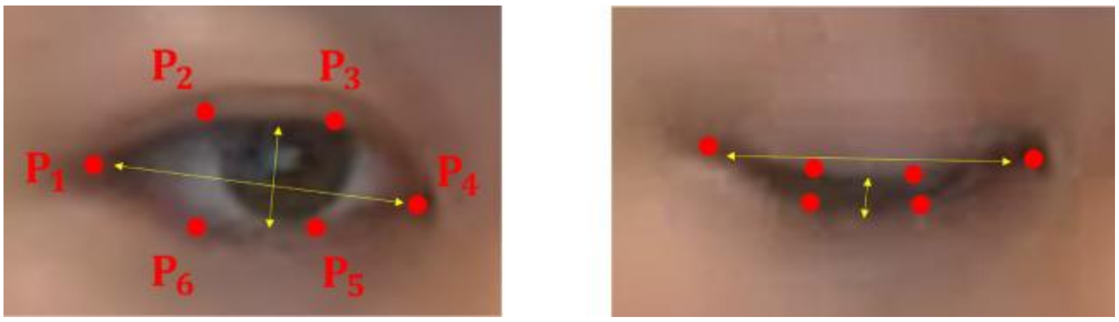


Figure 4.4 Eye Aspect Ratio (EAR)

Formula:

$$\text{EAR} = \frac{\|P2 - P6\| + \|P3 - P5\|}{2\|P1 - P4\|}$$

4.4.1 Making a Haar Cascade Classifier

The algorithm can be explained in four stages:

- Calculating Haar Features
- Creating Integral Images
- Using Adaboost
- Implementing Cascading Classifiers

It is important to remember that this algorithm requires a lot of positive images of faces and negative images of non-faces to train the classifier, similar to other machine learning models.

4.4.2 Calculating Haar Features

A Haar feature is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. The types of haar features is shown in Figure 4.5.

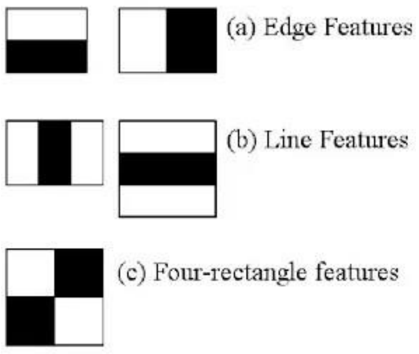


Figure 4.5 Types of Haar features

These features can be difficult to determine for a large image. This is where integral images come into play because the number of operations is reduced using the integral image.

4.4.3 Creating Integral Images

Integral images essentially speed up the calculation of these Haar features as shown in the below Figure 4.6. Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the Haar features.

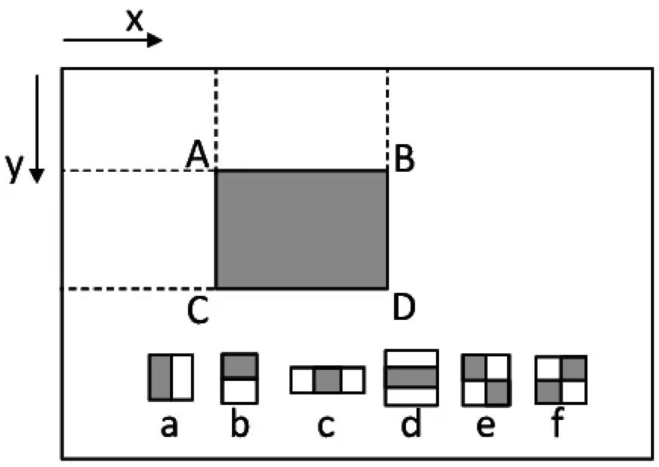


Figure 4.6 Working of an integral images

It's important to note that nearly all of the Haar features will be irrelevant when doing object detection, because the only features that are important are those of the object.

4.4.4 Adaboost Training

Adaboost essentially chooses the best features and trains the classifiers to use them. It uses a combination of "weak classifiers" to create a "strong classifier" that the algorithm can use to detect objects as shown in Figure 4.7

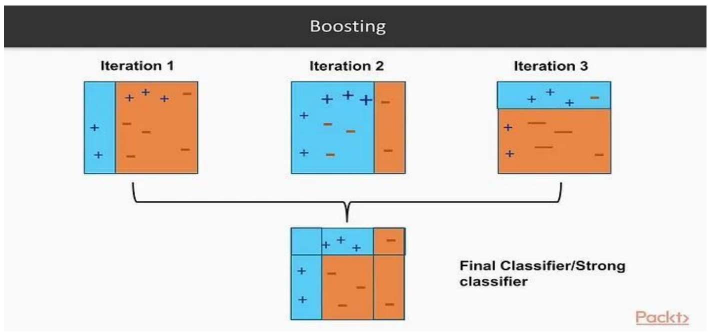


Figure 4.7 Representation of a boosting algorithm

The weak learners are created by moving a window over the input image, and computing Haar features for each subsection of the image. This difference is compared to a learned threshold that separates non-objects from objects. Because these are "weak classifiers," a large number of Haar features is needed for accuracy to form a strong classifier. The last step combines these weak learners into a strong learner using cascading classifiers.

4.4.5 Implementing Cascade Classifiers

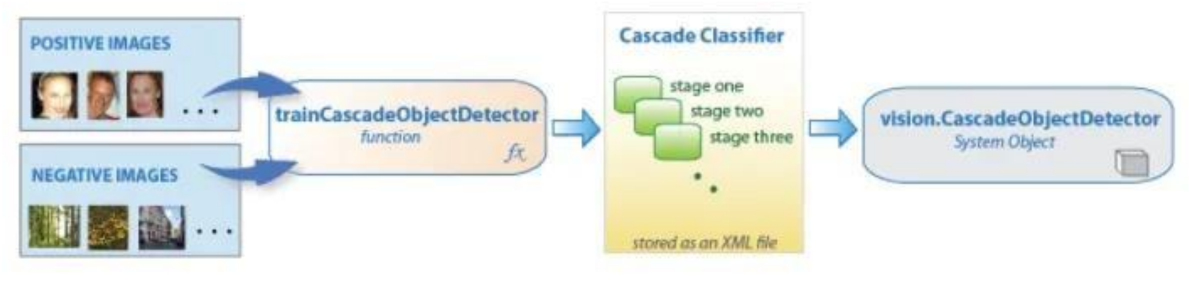


Figure 4.8 Flowchart of cascade classifiers

The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners. The flowchart of cascade classifiers is shown in the above Figure 4.8.

Based on this prediction, the classifier either decides to indicate an object was found (positive) or move on to the next region (negative). Stages are designed to reject negative samples as fast as possible, because a majority of the windows do not contain anything of interest.

It's important to maximize a low false negative rate, because classifying an object as a non-object will severely impair your object detection algorithm. A video below shows Haar cascades in action. The red boxes denote "positives" from the weak learners.

Haar cascades are one of many algorithms that are currently being used for object detection. One thing to note about Haar cascades is that it is very important to reduce the false negative rate, so make sure to tune hyper-parameters accordingly when training your model.

4.5 SYSTEM ARCHITECTURE OF CNN

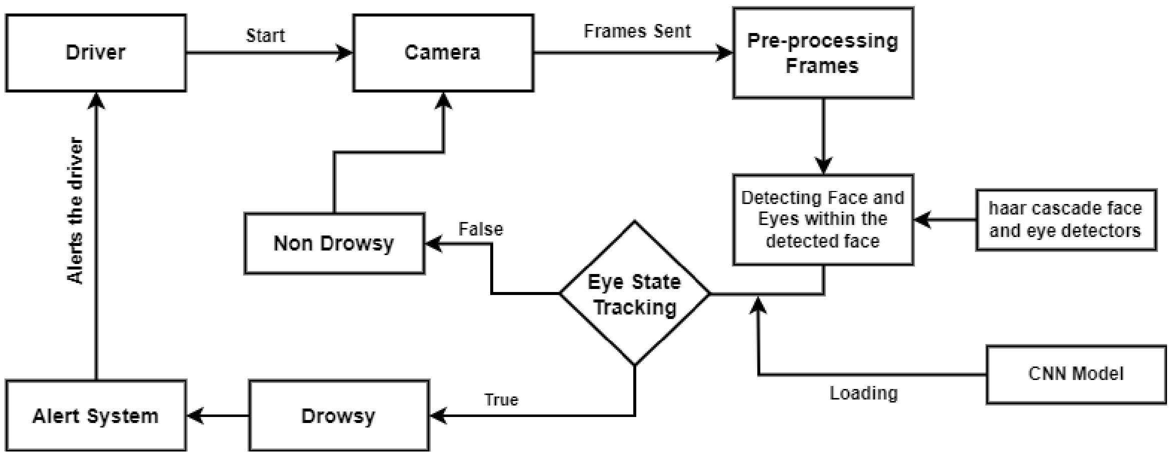


Figure 4.9 Proposed Architecture

4.6 WORKING OF CNN

Proposed System Architecture of CNN is shown in the figure 4.9. In this project, in order to reduce the accidents that are caused by driver's drowsiness or sleepiness, a driver drowsiness detection system is developed with the help of deep learning techniques. Initially Dataset contains images of both the closed and open eyes of several persons were collected to train the model. Before the training of model takes place, the images are preprocessed so that the dataset can be directly applied to the deep learning algorithm for further process. After preprocessing, our model has been trained. A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. Our system will be installed in automobiles such as cars, buses etc. Our system continuously monitors the eyes of the driver. Whenever the user feels drowsy (He / She feels sleepy by closing the eyes), our system alerts the driver with a beep sound as its primary response. The Alarm goes on increasing when the driver doesn't wakeup. Then our system is making a phone call to the driver's emergency contact person or relatives to inform that the person feels drowsy or sleeping while driving along with the driver’s coordinates. Thus, this system effectively reduces the risk of road accidents that are caused by driver’s carelessness.

4.7 MODULE DESCRIPTION

- Data Collection
- Building and training a CNN Model
- Face and Eye Detection
- Drowsiness Detection
- Alert System

4.7.1 Data Collection

The proposed system used image dataset of eyes of persons in both open and closed state. The ability to acquire data can make or break the solution. Not only is getting data usually the most important part of a deep learning project, but it’s also often the hardest. Luckily, there are many potential data sources.

Publicly Available Datasets

The best source of data is often publicly available datasets. Sites like Kaggle host thousands of large, labelled data sources. Working with these curated datasets helps reduce the overhead of starting a deep learning project.

Web scraping/APIs

Online news, social media posts, and search results represent rich streams of data, which can leverage for our deep learning projects. This is done via Web scraping: the extraction of data from websites. APIs can be used to gather data from different applications. While some APIs are free, others are paid services.

4.7.2 Preprocessing

Once the dataset is built, it needs to preprocess into useful features for our deep learning models. At a high-level, the primary goals in preprocessing data for neural networks:

1) clean our data

2) handle categorical features and text

3) scale our real-valued features using normalization or standardization techniques.

Cleaning data

Often, the datasets contain noisy examples, extra features, missing data and outliers. It is good practice to test for and remove outliers, remove unnecessary features, fill-in missing data, and filter out noisy examples.

Handling categorical data and text

Neural networks expect numbers as their inputs. Convert all categorical data and text to real-valued numbers. Categorical variables are handled by assigning each option its own unique integer or converting them to one-hot encodings. - When working with strings of raw text, few extra processing steps is need to be handled before encoding our words as integers. These steps include tokenizing our data (splitting our text into individual words/tokens), and padding our data (adding padding tokens to make all of our examples the same length).

Scaling features

Because neural networks is initialized with small weights to stabilize training, our models will struggle when faced with input features that have large values. As a result, real-valued features is scaled in two ways: Features are normalized so that they are between 0 and 1, and standardize them so they have a mean of zero and a variance of one.

4.7.3 Building and Training a CNN Model

Once the pre-processing of the datasets is done, CNN need to be built and trained by splitting and balancing the dataset into trained data and test data.The data is split in a balance way for training and testing.

Splitting and Balancing the Dataset

Once the data have been processed , it's time to split our dataset. Generally, the data is split into two datasets: training and validation. In certain cases, a third holdout dataset is created , referred to as the test set.

Splitting Our Data

10-30% of the data is saved for validation and testing. When there are smaller corpus, it is more important to assign a larger proportion of data to the validation set. This helps ensure that our validation dataset better represents the true distribution of our data.

Scikit-learn provides the train_test_split function, which splits our data into training and validation datasets and specifies the size of our validation data.

Stratified Train-Test Splits

It is possible that more instances of our minority classes end up in either the training or the validation set. In the first case, our validation metrics will not accurately capture our model's ability to classify the minority class. In the second case, the model will overestimate the probability of the majority class.

The solution is to use a stratified split: a split that ensures the training and validation sets have the same proportion of examples from each class.

If the train_test_split function's stratify parameter is set to our array of labels, the function will compute the proportion of each class, and ensure that this ratio is the same in our training and validation data.

4.7.4 Face Detection

Face detection is the process of detecting and locating human faces within an image or video stream. Two popular techniques for face detection are using a Convolutional Neural Network (CNN) model and using a Haar Cascade classifier.

To use a CNN model for face detection, the model is first trained on a large dataset of images that contain faces. Once trained, the model can then be used to detect faces in new images by passing the image through the network and looking for patterns that resemble faces.

Haar Cascade algorithm works by sliding a window over the image and calculating a score for each window based on the presence of certain features. If the score exceeds a certain threshold, the window is classified as containing an object. In the case of face detection, the Haar Cascade classifier is trained on a large dataset of images that contain faces, and the algorithm learns to recognize patterns such as the arrangement of eyes, nose, and mouth.

Both CNN models and Haar Cascade classifiers have their advantages and disadvantages in face detection. CNN models are often more accurate and can detect faces in more challenging lighting conditions or orientations. However, they can be computationally expensive and require a lot of training data.

Here, CNN is used for model building and using that model, face is detected in the frame obtained by camera using haar cascade files.

4.7.5 Eye Detection

The eyes is detected in the frame by using same technique as face detection. Using the closed and open eyes image dataset, the model is trained and detecting the eyes in the detected face using haar cascade eye files.

4.7.6 Drowsiness Detection

A drowsiness detection module can play a critical role in detecting when a driver is becoming drowsy or falling asleep at the wheel. The module typically utilizes a camera or other sensors to monitor the driver's physical behaviour and detect signs of drowsiness.

One common approach to implementing a drowsiness detection module is to use a machine learning algorithm, such as a convolutional neural network (CNN), to analyze video or image data of the driver's face and eyes. The algorithm can be trained on a large dataset of labeled data, which includes both drowsy and alert states, to learn to recognize the patterns that indicate drowsiness, such as drooping eyelids, head nodding, or yawning.

Once the algorithm detects that the driver is becoming drowsy, it can trigger an alert to the driver, such as an audible alarm, vibration, or visual cue. The alert can prompt the driver to take a break, switch drivers, or take other measures to avoid falling asleep at the wheel.

Overall, a drowsiness detection module can be an important component of a driver drowsiness detection project, as it can help prevent accidents caused by drowsy driving. The module can be integrated with other safety features, such as lane departure warning systems or collision avoidance systems, to provide a comprehensive safety solution for drivers.

4.7.7 Alert System

An alert system is a crucial component of a driver drowsiness detection project, as it can help prevent accidents caused by drowsy driving. When the drowsiness detection module detects that the driver is becoming drowsy or falling asleep at the wheel, the alert system can trigger an alert to the driver, prompting them to take action.

There are several types of alert systems that can be used in a driver drowsiness detection project, including:

- **Audible alarms:** These are sounds, such as beeps or buzzers that alert the driver when they are becoming drowsy.
- **Visual cues:** These are visual warnings, such as flashing lights or icons on the dashboard, that alert the driver when they are becoming drowsy.

- **Voice alerts:** These are spoken warnings, such as "wake up" or "pull over," that alert the driver when they are becoming drowsy.
- **Alerting the emergency contact persons:** A phone call can be made to the driver’s emergency contact person or relative to alert them with the exact location of the driver.

4.8 SUMMARY

This chapter concludes the proposed architecture and details about the working of the proposed system. The description of the modules are explained in brief along with the components of drowsiness detection system.

CHAPTER 5

SOFTWARE DESCRIPTION

5.1 INTRODUCTION

The purpose of the Software Requirement Specification is to produce the specification of the analysis task and also to establish complete information about the requirement, behaviour and also the other constraint like functional performance and so on. The main aim of the Software Requirement Specification is to completely specify the technical requirements for the software product in a concise and in unambiguous manner

5.2 SYSTEM SPECIFICATION

Software Used:

➢ Operating System	:	Windows 7 / 8/ 10
➢ Language	:	Python
➢ IDE	:	Anaconda, Notebook

Hardware Used:

➢ Processor	:	Ryzen5
➢ Ram	:	16 GB
➢ Hard Disk	:	120 GB

5.3 ANACONDA

Anaconda (Python distribution)

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS.

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.
The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code

Conda

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

Anaconda Cloud

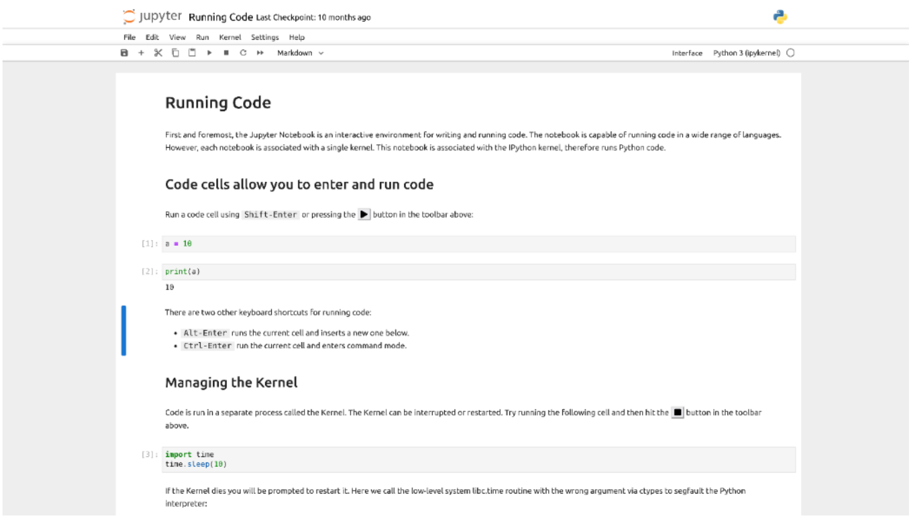
Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages. Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications.

5.4 JUPYTER NOTEBOOK

As a web application in which you can create and share documents that contain live code, equations, visualizations as well as text, the Jupyter Notebook is one of the ideal tools to help you to gain the data science skills you need.

What is a Jupyter Notebook?

In this case, "notebook" or "notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations, ... Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description, and its results, as well as, they can be executed perform the data analysis in real time.



jupyter Running Code Last Checkpoint: 10 months ago
File Edit View Run Kernel Settings Help
Markdown
Interface Python 3 (ipykernel)

Running Code

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefore runs Python code.

Code cells allow you to enter and run code

Run a code cell using `Shift-Enter` or pressing the ▶ button in the toolbar above:

```
[1]: a = 10
```

```
[2]: print(a)
10
```

There are two other keyboard shortcuts for running code:

- `Alt-Enter` runs the current cell and inserts a new one below.
- `Ctrl-Enter` run the current cell and enters command mode.

Managing the Kernel

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted. Try running the following cell and then hit the ■ button in the toolbar above.

```
[3]: import time
     time.sleep(10)
```

If the Kernel dies you will be prompted to restart it. Here we call the low-level system libc.time routine with the wrong argument via ctypes to segfault the Python interpreter:

Figure 5.1 Jupyter Notebook

Installation

Use a handy tool that comes with Python called pip to install Jupyter Notebook like this: $ pip install jupyter

The next most popular distribution of Python is Anaconda. Anaconda has its own installer tool called conda that you could use for installing a third-party package. However, Anaconda comes with many scientific libraries preinstalled, including the Jupyter Notebook, so you don’t actually need to do anything other than install Anaconda itself.

Exporting Notebooks

When you are working with Jupyter Notebooks, you will find that you need to share your results with non-technical people. When that happens, you can use the nbconvert tool which comes with Jupyter Notebook to convert or export your Notebook into one of the following formats:

- HTML
- LaTeX
- PDF
- RevealJS
- Markdown
- Executable script

5.5 PYTHON IDLE 3.6.6

- In this project, Python IDLE 3.6.6 is also used as an IDE
- Python IDLE offers a full-fledged file editor, which gives you the ability to write and execute Python programs from within this program. The built-in file editor also includes several features, like code completion and automatic indentation that will speed up your coding workflow.

5.6 PACKAGES USED

The pacakages for the module to be imported are as follows:

5.6.1 cv2

cv2 (opencv-python) is a Python library that allows you to perform image processing and computer vision tasks. It provides a wide range of features, including object detection, face recognition, and tracking.

5.6.2 keras

keras is a deep learning API written in Python, running on top of the machine learning platform tensorflow. It was developed with a focus on enabling fast experimentation and providing a delightful developer experience.

5.6.3 tensorflow

tensorflow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

5.6.4 numpy

numpy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. numpy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. numpy stands for Numerical Python.

5.6.5 imutils

contours, detecting edges, and much more easier with OpenCV and both Python 2.7 and Python 3.

5.6.6 matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, pan, update.

5.6.7 os

The os module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

5.7 SUMMARY

This chapter concludes the description of software requirements, the platform in which the code is written. It also briefs about the packages imported for the working of the modules in drowsiness detection.

CHAPTER 6

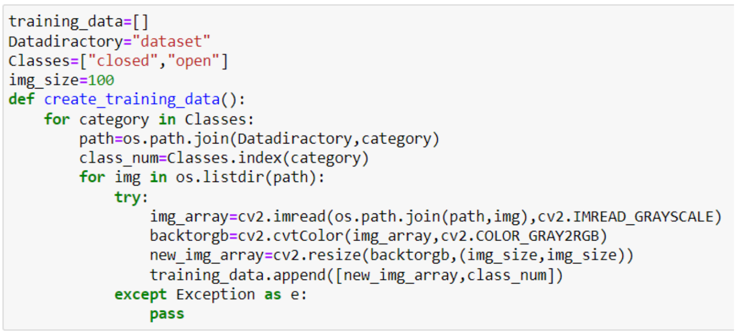
RESULTS AND DISCUSSION

6.1 INTRODUCTION

This chapter discuss about the results obtained from the implemented deep learning model. It comprises of the preprocessing of the image datasets, creating the CNN model, training and saving the model. The graph comparing the loss and epoch, accuracy and epoch is also discussed in this chapter.

6.2 RESULTS OF THE WORK

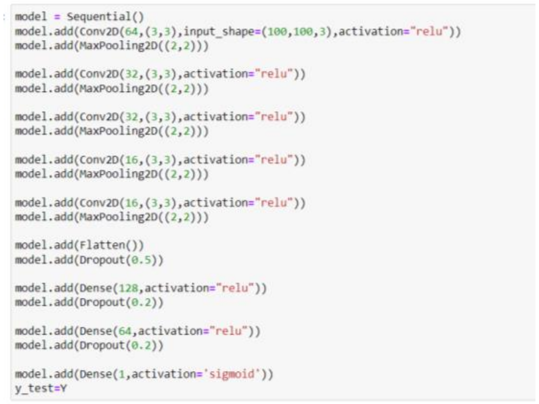
First, preprocessing of the image dataset is done by defining the create_training_data() and creating directory of 2 classes as in Figure 6.1.



```
training_data=[]
Datadiractory="dataset"
Classes=["closed","open"]
img_size=100
def create_training_data():
    for category in Classes:
        path=os.path.join(Datadiractory,category)
        class_num=Classes.index(category)
        for img in os.listdir(path):
            try:
                img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
                backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
                new_img_array=cv2.resize(backtorgb,(img_size,img_size))
                training_data.append([new_img_array,class_num])
            except Exception as e:
                pass
```

Figure 6.1 Preprocessing of the data

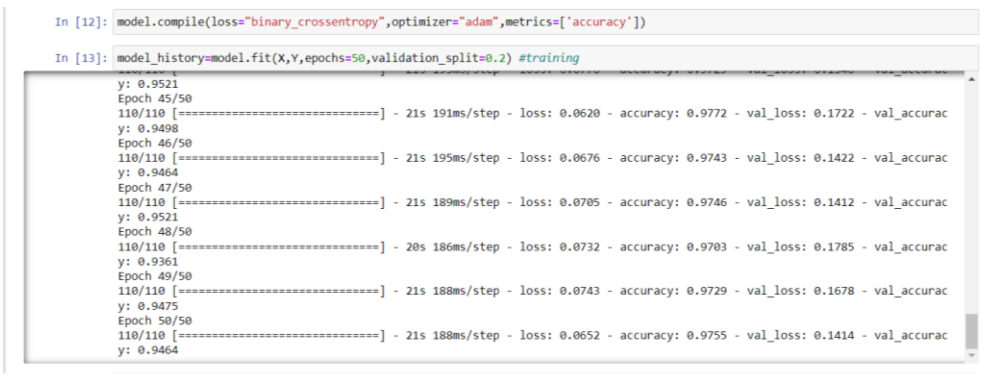
A batch size as 32 and target size as (100, 100) is taken. Then, our own CNN model is created with a dense layer as hidden layer as shown in Figure 6.2.



```
model = Sequential()
model.add(Conv2D(64,(3,3),input_shape=(100,100,3),activation="relu"))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(32,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(32,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(16,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(16,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(128,activation="relu"))
model.add(Dropout(0.2))

model.add(Dense(64,activation="relu"))
model.add(Dropout(0.2))

model.add(Dense(1,activation='sigmoid'))
y_test=Y
```

Figure 6.2 Defining the model structure

Training the model for different number of epoch increases the accuracy of the model. Epoch of 10,20,50 were done to improve the accuracy as in Figure 6.3

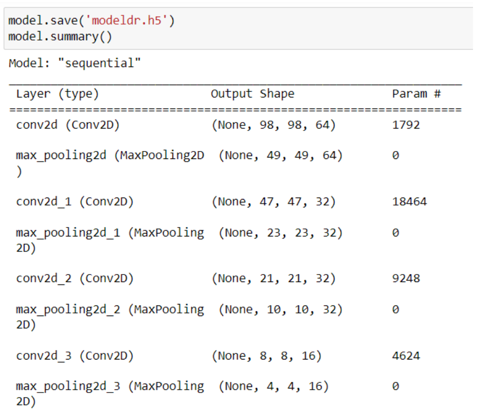


```
In [12]: model.compile(loss="binary_crossentropy",optimizer="adam",metrics=['accuracy'])

In [13]: model_history=model.fit(X,Y,epochs=50,validation_split=0.2) #training
```

```
y: 0.9521
Epoch 45/50
110/110 [==============================] - 21s 191ms/step - loss: 0.0620 - accuracy: 0.9772 - val_loss: 0.1722 - val_accurac
y: 0.9498
Epoch 46/50
110/110 [==============================] - 21s 195ms/step - loss: 0.0676 - accuracy: 0.9743 - val_loss: 0.1422 - val_accurac
y: 0.9464
Epoch 47/50
110/110 [==============================] - 21s 189ms/step - loss: 0.0705 - accuracy: 0.9746 - val_loss: 0.1412 - val_accurac
y: 0.9521
Epoch 48/50
110/110 [==============================] - 20s 186ms/step - loss: 0.0732 - accuracy: 0.9703 - val_loss: 0.1785 - val_accurac
y: 0.9361
Epoch 49/50
110/110 [==============================] - 21s 188ms/step - loss: 0.0743 - accuracy: 0.9729 - val_loss: 0.1678 - val_accurac
y: 0.9475
Epoch 50/50
110/110 [==============================] - 21s 188ms/step - loss: 0.0652 - accuracy: 0.9755 - val_loss: 0.1414 - val_accurac
y: 0.9464
```

Figure 6.3 Training the model

Saving the Model in a h5 file in the name ‘modeldr.h5’ is done to contain the trained data model to use it for real time prediction as in Figure 6.4.

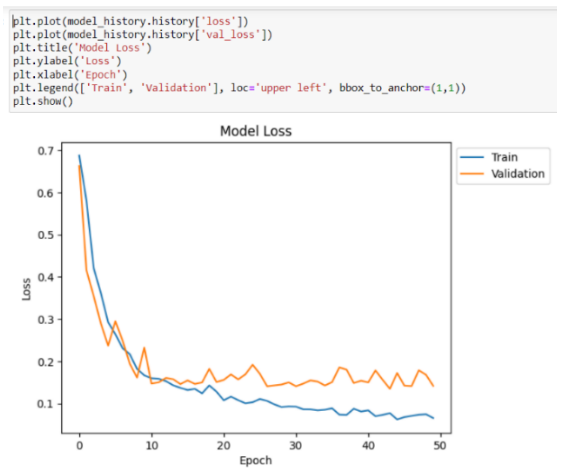


```
model.save('modeldr.h5')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 49, 49, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 47, 47, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 23, 23, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 21, 21, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 10, 10, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 8, 8, 16)          4624

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 16)         0
 2D)
```

Figure 6.4 Saving the Model

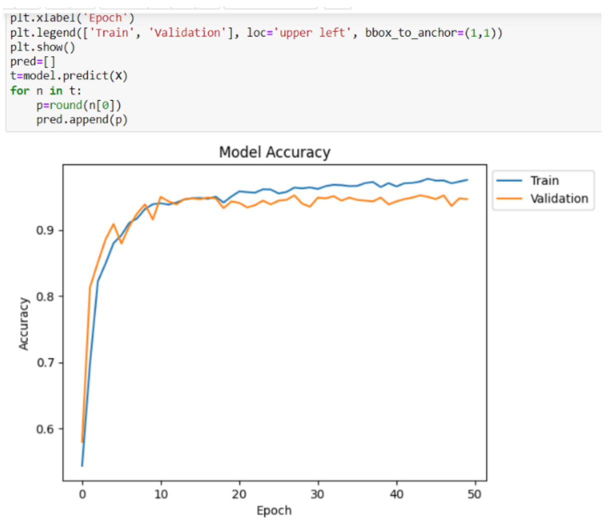
Figure 6.5 shows the decrease in loss during epochs increase while training



```
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()
```

Figure 6.5 Epochs vs Loss Graph

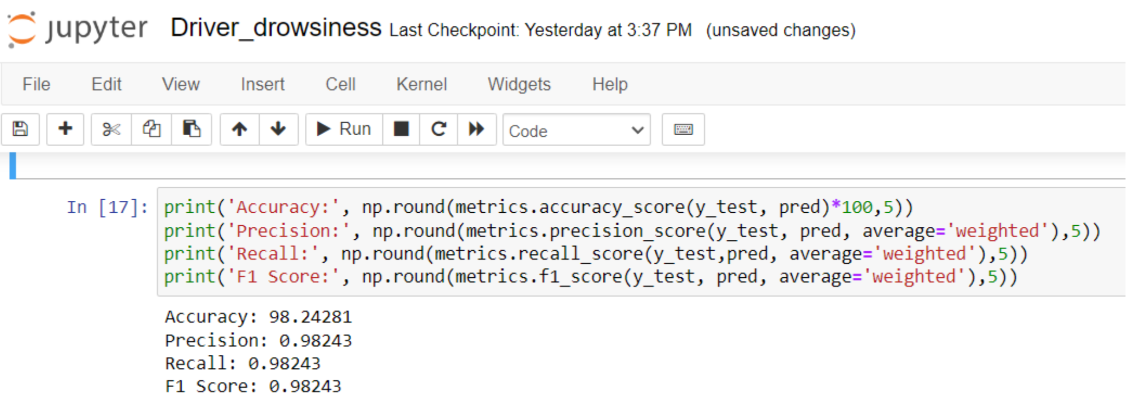
Figure 6.6 shows the decrease in loss during epochs increase while training



```
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()
pred=[]
t=model.predict(X)
for n in t:
    p=round(n[0])
    pred.append(p)
```

Figure 6.6 Epochs vs Accuracy Graph

Performance Metrics such as Accuracy, Precision, Recall, F1 score is calculated as shown in Figure 6.7.



jupyter Driver_drowsiness Last Checkpoint: Yesterday at 3:37 PM (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

```
In [17]: print('Accuracy:', np.round(metrics.accuracy_score(y_test, pred)*100,5))
         print('Precision:', np.round(metrics.precision_score(y_test, pred, average='weighted'),5))
         print('Recall:', np.round(metrics.recall_score(y_test,pred, average='weighted'),5))
         print('F1 Score:', np.round(metrics.f1_score(y_test, pred, average='weighted'),5))

         Accuracy: 98.24281
         Precision: 0.98243
         Recall: 0.98243
         F1 Score: 0.98243
```

Figure 6.7 Performance Analysis

6.3 EVALUATION

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses. Model evaluation is important to assess the efficacy of a model during initial research phases, and it also plays a role in model monitoring.

6.3.1 r2 Score

The r2 score varies between 0 and 100%. The proportion of the variance in the dependent variable that is predictable from the independent variable(s). Another definition is (total variance explained by model) / total variance. So if it is 100%, the two variables are perfectly correlated, i.e., with no variance at all.

- ✓ r2 score for our project is 0.9383.

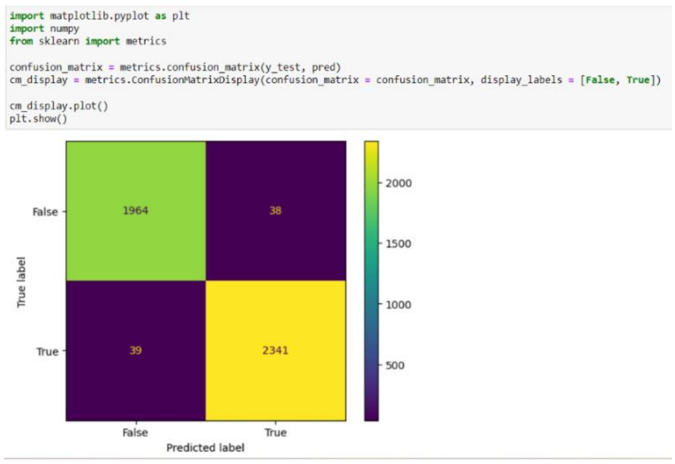
6.3.2 Accuracy

Accuracy score is used to measure the model performance in terms of measuring the ratio of sum of true positive and true negatives out of all the predictions made.

- ✓ Accuracy of our model is about 98 %.

6.3.3 Confusion Matrix

A confusion matrix is a table that is used to define the performance of a classification algorithm as shown in Figure 6.8. A confusion matrix visualizes and summarizes the performance of a classification algorithm.



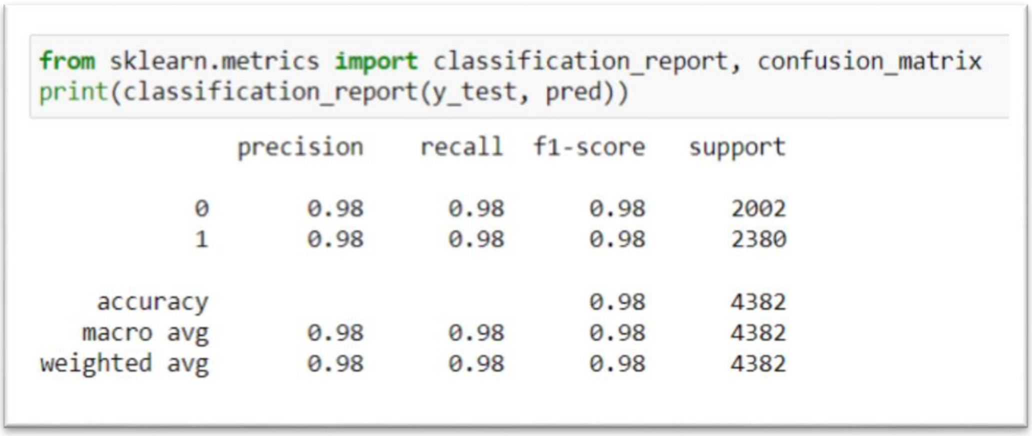
```python
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(y_test, pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

cm_display.plot()
plt.show()
```

Figure 6.8 Confusion Matrix

TP = 1964, FP = 38, FN = 39, TN = 2341

6.3.4 Classification Report

A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of your trained classification model as shown in Figure 6.9.



```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2002
           1       0.98      0.98      0.98      2380

    accuracy                           0.98      4382
   macro avg       0.98      0.98      0.98      4382
weighted avg       0.98      0.98      0.98      4382
```

Figure 6.9 Classification Report

6.4 FINDINGS

In this project, a driver drowsiness detection system using convolutional neural networks (CNN) and OpenCV have been developed. The system is designed to detect the drowsiness level of the driver in real-time and alert them if they are falling asleep while driving.

The existing systems for drowsiness detection use simple rules-based algorithms that are not very accurate and do not work well in different lighting conditions. Therefore, more advanced system that uses deep learning techniques to improve accuracy have been developed.

Our system uses a CNN model that is trained on a large dataset of images of drivers with varying levels of drowsiness. The model takes in live video feed from a camera mounted on the dashboard of the car and uses OpenCV to detect the face of the driver. It then analyses the facial features to determine the level of drowsiness. If the driver is found to be drowsy, the system alerts them with an alarm or a visual alert, depending on their preference.

Our model outperforms the existing systems in terms of accuracy and works well in different lighting conditions. It can also be easily integrated into existing car systems. This project aims to improve road safety by reducing the number of accidents caused by driver drowsiness.

6.5 SUMMARY

This chapter concludes about the results obtained from the CNN model. The performance metrics like r2 score, precision, recall, accuracy is briefly discussed. The classification report of the deep learning model is shown.

CHAPTER 7

CONCLUSION

When looking at the Viola-Jones algorithm and Convolutional Neural Networks for face detection, it is difficult to say which one is best overall. Both methods have strengths and weaknesses when it comes to certain areas of face detection. While CNNs are much faster, they do require more memory space and are therefore are more expensive to implement. The Viola-Jones algorithm is older but given its minimal memory requirement it is implemented much more easily. The next best thing would be to implement some features from the Viola-Jones algorithm along with the Convolutional Neural Networks. While there has been some talk of implementing CNNs using Viola-Jones' cascading feature, the issue of CNNs accessing too much RAM is a constant problem**.** The driver drowsiness detection system using CNN and OpenCV is an effective solution to improve road safety by alerting drivers who are at risk of falling asleep while driving. The system uses advanced deep learning techniques to accurately detect drowsiness levels in real-time and alert the driver accordingly. By leveraging the power of computer vision and machine learning, a solution have been developed that is more accurate and robust than traditional rule-based systems.

7.1 FUTURE SCOPE

In the coming future, in addition to the alarm system, a system can be developed which can access the vehicle's brake system, which can apply the brake of the vehicle in an emergency situation in any safe places like left side ends of the roads. Brakes cannot be applied in the centre lane itself. Our system will also detect the exact location of the vehicle by using GPS Module and alerts the driver's relation and even also the nearby hospital if it met with an accident.

APPENDIX

SOURCE CODE

MACHINE LEARNING

```
from scipy.spatial import distance
from imutils import face_utils
import imutils
import dlib
import cv2
import winsound
frequency = 2500
duration = 1000
def eye_aspect_ratio(eye):
        A = distance.euclidean(eye[1], eye[5])
        B = distance.euclidean(eye[2], eye[4])
        C = distance.euclidean(eye[0], eye[3])
        ear = (A + B) / (2.0 * C)
        return ear
thresh = 0.25
frame_check = 20
detect = dlib.get_frontal_face_detector()
predict = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
cap=cv2.VideoCapture(0)
flag=0
while True:
        ret, frame=cap.read()
        frame = imutils.resize(frame, width=450)
```

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
subjects = detect(gray, 0)
for subject in subjects:
        shape = predict(gray, subject)
        shape = face_utils.shape_to_np(shape)#converting to NumPy Array
        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]
        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)
        ear = (leftEAR + rightEAR) / 2.0
        leftEyeHull = cv2.convexHull(leftEye)
        rightEyeHull = cv2.convexHull(rightEye)
        cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
        cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
        if ear < thresh:
                flag += 1
                #print (flag)
                if flag >= frame_check:
                        cv2.putText(frame,"ALERT! ", (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
                        cv2.putText(frame, "ALERT!, (10,325),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
                        winsound.Beep(frequency,duration)
        else:
                flag = 0
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
if key == ord("s"):
        cv2.destroyAllWindows()
```

```
                cap.release()
                break
```

DEEP LEARNING

```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import
Dense,Dropout,Conv2D,MaxPooling2D,Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator

training_data=[]
Datadiractory="dataset"
Classes=["closed","open"]
img_size=100
def create_training_data():
   for category in Classes:
      path=os.path.join(Datadiractory,category)
      class_num=Classes.index(category)
      for img in os.listdir(path):
         try:
```

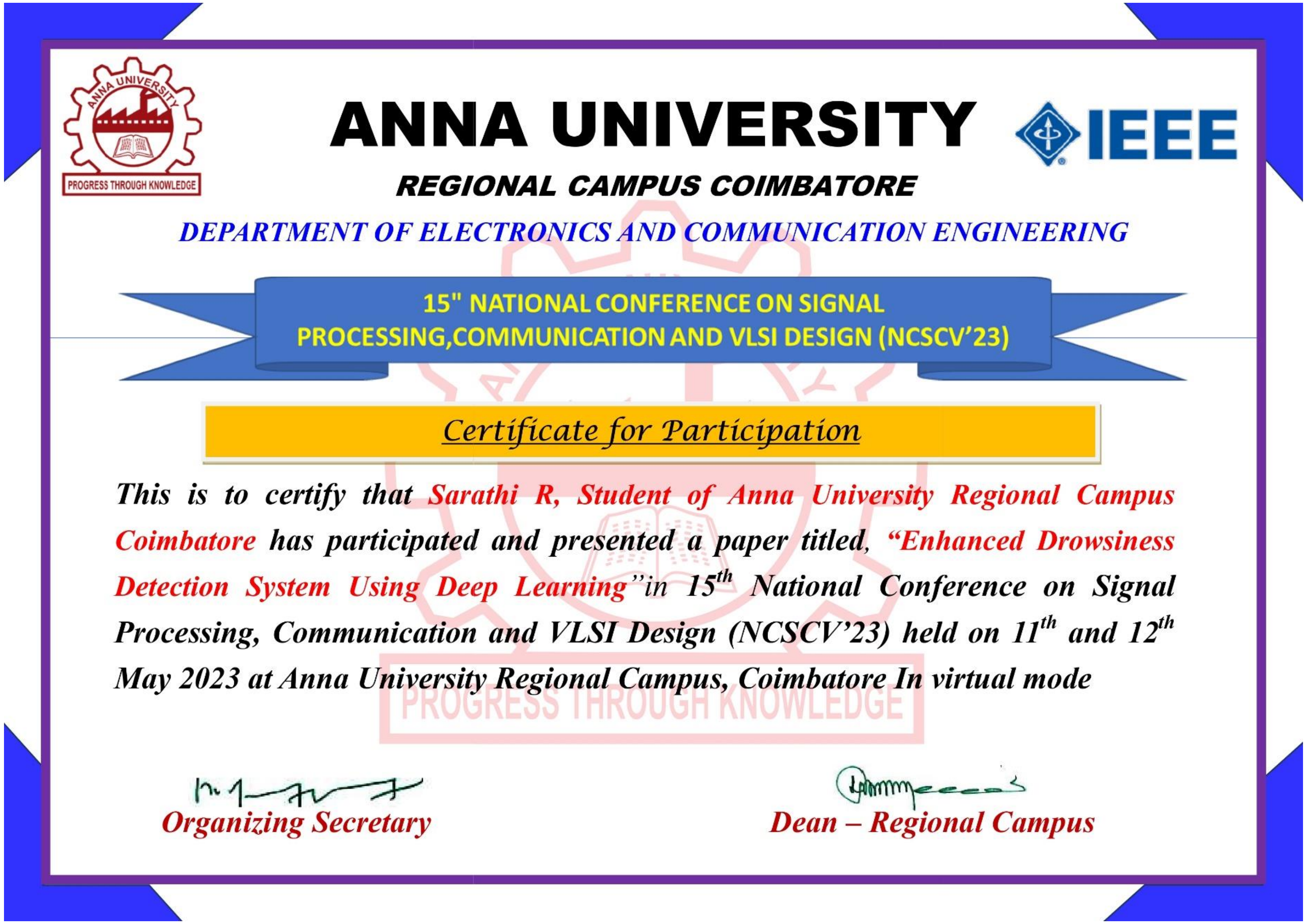
```
img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
                backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
                new_img_array=cv2.resize(backtorgb,(img_size,img_size))
                training_data.append([new_img_array,class_num])
            except Exception as e:
                pass

model = Sequential()
model.add(Conv2D(64,(3,3),input_shape=(100,100,3),activation="relu"))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(32,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(32,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(16,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(16,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(64,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1,activation='sigmoid'))
y_test=Y
model.compile(loss="binary_crossentropy",optimizer="adam",metrics =
['accuracy'])
```

```
model.save('modeldr.h5')


face = cv2.CascadeClassifier('haarcascade/haarcascade_frontalface_alt.xml')
leye = cv2.CascadeClassifier('haarcascade/haarcascade_eye.xml')
reye = cv2.CascadeClassifier('haarcascade/haarcascade_eye.xml')
eyes=cv2.CascadeClassifier('haarcascade\haarcascade_eye.xml')
lbl=['Closed eyes','Open eyes']
model = load_model('modeldr.h5')
path = os.getcwd()
cap = cv2.VideoCapture(0) #to access the camera
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
count=0
score=0
thicc=2
rpred=[99]
lpred=[99]
while(True):
    ret, frame = cap.read()
    height,width = frame.shape[:2]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = 
face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25)) 
#Face detection
    eye=eyes.detectMultiScale(gray)
    left_eye = leye.detectMultiScale(gray,1.1,4)
    right_eye =  reye.detectMultiScale(gray,1.1,4)


    cv2.rectangle(frame, (0,height-50) , (200,height) , (0,0,0) 
,thickness=cv2.FILLED)
    for (x,y,w,h) in faces:
```

```
    cv2.rectangle(frame, (x,y) , (x+w,y+h) , (150,150,150) , 1)
for (x,y,w,h) in eye:
    cv2.rectangle(frame, (x,y),(x+w,y+h) ,(150,150,150) , 1)
for (x,y,w,h) in right_eye:
    r_eye=frame[y:y+h,x:x+w]
    count=count+1
    r_eye = cv2.resize(r_eye,(100,100))
    r_eye= r_eye/255
    r_eye=  r_eye.reshape(100,100,-1)
    r_eye = np.expand_dims(r_eye,axis=0)
    rpred = model.predict(r_eye)
    if(rpred>=0.5):
        lbl='Open'
    else:
        lbl='Closed'
    break
for (x,y,w,h) in left_eye:
    l_eye=frame[y:y+h,x:x+w]
    count=count+1
    l_eye = cv2.resize(l_eye,(100,100))
    l_eye= l_eye/255
    l_eye=l_eye.reshape(100,100,-1)
    l_eye = np.expand_dims(l_eye,axis=0)
    lpred = model.predict(l_eye)
    if(lpred>=0.5):
        lbl='Open'
    else:
        lbl='Closed'
    break
    if(lbl=="Closed" ):
        score=score+1
        cv2.putText(frame,"Closed",(10,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)
    else:
        score=score-1
        cv2.putText(frame,"Open",(10,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)
    if(score<0):
        score=0
    cv2.putText(frame,'Score:'+str(score),(100,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)
    if(score>5):
        cv2.putText(frame,'DROWSY DRIVER.....',(100,100), font,
1,(0,0,255),1,cv2.LINE_AA)
        winsound.Beep(frequency,duration)
        if(thicc<16):
            thicc= thicc+2
        else:
            thicc=thicc-2
            if(thicc<2):
                thicc=2
        cv2.rectangle(frame,(0,0),(width,height),(0,0,255),thicc)
    cv2.imshow('Driver drowsiness detection',frame)
    if cv2.waitKey(1) & 0xFF == ord('s'):
        break
cap.release()
cv2.destroyAllWindows()
```

LIST OF PUBLICATIONS

1. ANANDHAPRIYA, R. JERUSHA JAISAL, J.L. MONISH, R. SARATHI, R. KANDASAMY, K. (2023) ‘ENHANCED DROWSINESS DETECTION SYSTEM USING DEEP LEARNING’ Presented the conference paper in proceedings of 15th National Conference on Signal Processing Communication and VLSI. Vol 1, No.1, pp. 380-384



ANNA UNIVERSITY

REGIONAL CAMPUS COIMBATORE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

15" NATIONAL CONFERENCE ON SIGNAL PROCESSING,COMMUNICATION AND VLSI DESIGN (NCSCV'23)

Certificate for Participation

This is to certify that Sarathi R, Student of Anna University Regional Campus Coimbatore has participated and presented a paper titled, "Enhanced Drowsiness Detection System Using Deep Learning"in 15th National Conference on Signal Processing, Communication and VLSI Design (NCSCV'23) held on 11th and 12th May 2023 at Anna University Regional Campus, Coimbatore In virtual mode

[illegible]

Organizing Secretary

[illegible]

Dean – Regional Campus

REFERENCES

1. Cong Yang, Zhenyu Yang, Weiyu Li, “FatigueView: A Multi-Camera Video Dataset for Vision-Based Drowsiness Detection” in IEEE Transactions on Intelligent Transportation Systems, Vol. 24, No. 1, Jan 2023, pp 233 – 246, doi = 10.1109/TITS.2022.3216017.
2. Duy-Linh Nguyen, Muhamad Dwisnanto Putro and Kang-Hyun Jo, “Driver Behaviours Recognizer Based on Light-Weight Convolutional Neural Network Architecture and Attention Mechanism” in IEEE Access, July 11 2022, pp 71019 - 71029, doi = 10.1109/ACCESS.2022.3187185.
3. Lew, M.; Sebe, N.; Huang, T.; Bakker, E.; Vural, E.; Cetin, M.; Ercil, A.; Littlewort, G.; Bartlett, M.; Movellan, J. “Drowsy driver detection through facial movement analysis.In Human-Computer Interaction”; Springer: Berlin, Germany, 2007; Volume 4796, pp. 6–18.
4. Mohamed Hedi Baccour, Franke Driewer, Tim Schack and Enkelejda Kasneci, “Comparative Analysis of Vehicle-Based and Driver-Based Features for Driver Drowsiness Monitoring by Support Vector Machines” in IEEE Transactions on Intelligent Transport Systems, Vol 23, No. 12, Dec 2022, pp 23164 – 23178, doi = 10.1109/TITS.2022.3207965.
5. Phillip Ian Wilsom, Dr John Fernandez, “Facial feature detection using Haar classifiers” Journal of Computing Sciences in Colleges, vol. 21, pp 287-295, researchgate, 2006
6. Pushkal Pandey, Monil Sharma, Prakhar Saxena and Rajendra Kumar Dwivedi, “ Driver Drowsiness Monitoring and Detection using Machine Learning ” in IDCIoT 2023, pp 421 - 426, doi = 10.1109/IDCIoT56793.2023.10053497.
7. Shan An, Xin Ma, Rui Song, Yibin li, “Face detection and recognition with surf for human-robot Interaction”. IEEE International Conference on Automation and Logistics, August 2009, Vol 12, pp 342-349.
8. Xiao, F.; Bao, C.Y.; Yan, F.S. “Yawning detection based on gabor wavelets and LDA” J. Beijing Univ. Technol. 2009, Vol 35, pp 409–413.
9. Yin, B.-C.; Fan, X.; Sun, Y.-F. “Multiscale dynamic features based driver fatigue detection.Int. J. Pattern Recogn”. Artif. Intell. 2009, Vol 23, pp 575–589.