**MODULE NAME: ARTIFICIAL NEURAL NETWORKS**

**MODULE CODE: 7088CEM**

**JAHNAVI PRIYA RATAKONDA**

**STUDENT ID: 14155136**

[**ratakondaj@uni.coventry.ac.uk**](mailto:ratakondaj@uni.coventry.ac.uk)

**TITLE NAME: DROWSINESS PREDICTION THROUGH IMAGE USING DEEP LEARNING**

**Word Count = 4351**

**ABSTRACT:**

In recent times, there has been a marked increase in the incidence of road accidents, leading to the tragic loss of precious human lives. This alarming trend has become all too common, causing great concern among the public and authorities alike. The severity of these accidents has been such that the impact has been nothing less than devastating, leaving behind a trail of shattered families and communities. One of the major causes of these accidents is driver fatigue, a state of reduced alertness and reaction time that can occur when an individual transitions from being awake to sleeping. Drowsy driving, which is driving while feeling tired or sleepy, is a significant factor contributing to collisions, injuries, and fatalities on the road. Despite the availability of some safety systems developed to enhance safety while driving, their usage is not widespread, as they are mainly found in high-end vehicles. To address this issue, a system has been designed to detect drowsiness based on four classes: the state of the eyes being fully open, fully closed, yawning, or not yawning. Drowsy driving poses a significant risk to road safety and can impact anyone who operates a motor vehicle, significantly increasing the likelihood of accidents. The statistics indicate a concerning number of injuries and fatalities every year due to drowsy driving. The drowsiness detection system employs convolutional neural networks (CNNs) in conjunction with two popular deep learning architectures, namely VGG (Visual geometry group) and MobileNetV2. The system processes an image input to detect the signs of drowsiness exhibited by the subject accurately. The system's accuracy and efficiency make it a vital tool in mitigating the risks associated with drowsy driving. Developing and deploying such systems can significantly enhance road safety and reduce the fatalities and injuries caused by road accidents.

**INTRODUCTION:**

Driver fatigue and drowsiness significantly contribute to car accidents, accounting for about 42%, as per recent studies. Although safety systems have been designed to detect driver drowsiness, they are primarily available in luxury cars and are not widely used [1]. However, intelligent systems can accurately perform complex tasks with advancements in deep learning and machine learning techniques. Deep learning algorithms are gaining popularity in detecting image features in real-life scenarios. Still, their significant size makes deploying them in real-time applications challenging due to high memory consumption and delayed response, which can negatively impact user experience. Driver drowsiness detection is an application that requires high accuracy and speed. To tackle this challenge, there is a need for deep learning models that not only have high accuracy but also have lower memory requirements and faster response times. Developing such models will enable the utilization of deep learning models for solving the problem of driver drowsiness detection in real-time scenarios. Such models can save countless lives on the road and make driver drowsiness detection accessible to all drivers. In summary, developing low-memory and high-accuracy deep-learning models for driver drowsiness detection will significantly reduce car accidents caused by driver fatigue or drowsiness. Additionally, with the broader availability of such models, the risk of accidents caused by driver drowsiness can be significantly reduced, making our roads safer for everyone. This system is designed to detect drowsiness in drivers by analyzing their facial expressions. It can determine whether a person's eyes are open or closed, whether they are yawning, or displaying any other symptoms of fatigue. Considering all these factors, the system can accurately predict when a driver is becoming drowsy and may be at risk of falling asleep at the wheel.

* **ANN (Artificial Neural Network):**

Artificial Neural Networks (ANNs) are a set of interconnected nodes or artificial neurons capable of processing and transmitting signals while learning from examples. They can be utilized for various tasks, such as classification, regression, clustering, dimensionality reduction, etc. Each artificial neuron receives signals, processes them, and can signal neurons connected to it. The signal at a connection is a weight, and the output of each neuron is calculated by a non-linear function of the sum of its inputs. Neurons and edges have a weight that adjusts as learning proceeds, increasing or decreasing the signal's strength at a connection. Some neurons may have a threshold that a signal is sent only if the aggregate signal crosses that threshold. Neural networks are usually trained using empirical risk minimization, and gradient-based methods such as backpropagation are used to estimate the network's parameters. As with all models trained with empirical risk minimization, such systems learn to perform tasks by considering examples without being manually programmed.

* **DEEP LEARNING:**

Training a deep learning model involves using a vast dataset and optimizing the neural network's parameters (weights and biases) through backpropagation. The model learns, through training, to identify patterns and features in the data, allowing it to predict or classify new data accurately. Deep learning is a subset of machine learning that focuses on training artificial neural networks to perform tasks without explicit programming. This approach is inspired by the structure and function of the human brain, where interconnected neurons work together to process information. Deep learning has been particularly successful in domains such as computer vision and image prediction, owing to its unparalleled ability to extract complex features from data and perform highly accurate predictions.

**RELATED WORK:**

* It is an unfortunate fact that driver drowsiness is one of the leading causes of road accidents, second only to driving while intoxicated. According to the National Highway Traffic Safety Administration (NHTSA), an estimated 100,000 accidents occur yearly in the United States due to driver fatigue. Shockingly, in 2013 alone, NHTSA reported that 72,000 accidents, 44,000 injuries, and 800 deaths were caused solely by driver drowsiness. The impact of drowsiness on a driver's reaction time and concentration cannot be overstated. When drivers feel fatigued, their reaction time to any situation becomes severely compromised, putting themselves and others on the road in great danger [5].
* Driver drowsiness and fatigue are leading causes of vehicular accidents. Developing a robust fatigue detection system poses a complex and informative challenge. OpenCV has developed a driver alert control system in the automotive sector that warns drivers when they become drowsy. This system utilizes a vehicle-mounted camera with a lane departure warning system. Similarly, Bosch has introduced a fatigue detection system that makes decisions based on data obtained from a sensor located on the steering wheel, vehicle speed, turn signals, and a front-mounted camera used for lane assist [4].
* As per the reports published by the National Crime Records Bureau (NCRB) in 2020, reckless driving and road accidents are the leading causes of increasing traffic accidents. The significance of getting enough sleep cannot be overstated, as human beings need to function correctly. However, many accidents and injuries occur due to the dangerous choices made by tired drivers and passengers. Driver drowsiness can be detected by analyzing various indicators such as eye movements, frontal and lateral head positions, and yawning [7].
* A recent study has utilized a deep learning approach to detect the sleep states of drivers in a driving environment. A convolutional neural network (CNN) model has been proposed to identify whether a driver's eyes are closed or open by analyzing constant face images. The potential applications of this model are vast, including but not limited to human-computer interface design, facial expression recognition, and driver fatigue-sleepiness determination. The proposed model has been trained on driver sleepiness data and tested on 4,846 authentic eye images from the Closed Eyes In The Wild (CEW) database. Moreover, various commonly used CNN models have been applied to the same data to compare the performance of the proposed model [8].
* The objective of this article is to introduce a state-of-the-art alarm system for detecting drowsiness in drivers, which is based on deep learning frameworks using f-RCNN and CNN. The system has been designed as a wireless, sensor-free, distraction-less, and vision-based solution to accurately monitor drivers' drowsiness. The system incorporates an automated eye region detection and eye states classification model developed using deep learning frameworks. The eye region is detected using f-RCNN, while the eye states are identified and classified using CNN. The system's prime contributions include developing a microcontroller-based alarm system with various intensities of drowsiness levels, which provides instantaneous remedies. We have incorporated a new dataset containing blink analysis and angular views (60 degrees right left) of eyes under various driving environments to make the system robust and effective. This system can significantly improve driver safety and reduce the risk of accidents caused by drowsiness [2].
* Drowsy driving is a primary concern for road safety, and the use of machine learning for driver drowsiness detection is an important research area. Several studies have been conducted on this topic, proposing various approaches based on features and machine learning algorithms. These features include eye movements, facial expressions, heart rate variability, and EEG signals. The machine learning algorithms range from traditional techniques like support vector machines and decision trees to more advanced approaches such as convolutional neural networks and artificial neural networks. These studies have produced remarkable results, with accuracy rates ranging from 94% to 97.5%, thereby demonstrating the efficacy of these approaches in detecting driver drowsiness [3].
* The previous research focuses on designing an efficient Human Machine Interface (HMI) system that utilizes deep learning techniques for driver monitoring. This innovative technology eliminates the need for hardware integration and instead uses deep learning concepts to monitor a driver's eye movement. Whenever the system detects fatigue, it promptly notifies the driver by triggering a buzzer. Additionally, it has an automatic feature that reduces the gear motor speed when drowsiness is detected. The deep learning algorithm employed in this study is the Convolutional Neural Network (CNN), which performs many tasks, including alert generation, image classification, feature extraction, and detection. This sophisticated technology is designed to improve driver safety, reduce the risk of accidents caused by driver fatigue, and enhance overall road safety [10].
* According to recent research reports, there are two primary ways of detecting driver drowsiness: subjective and objective methods. The personal approach involves asking the driver to complete a questionnaire, while the objective approach is monitoring the driver's behavior. However, the subjective method is less effective than the objective one in detecting drowsiness. The accurate method employs various techniques to detect driver drowsiness, such as monitoring eye movements, steering patterns, and vehicle speed. Based on recent research, the objective method is the best way to detect driver drowsiness. By using advanced technology, objective methods can provide accurate and reliable results, helping to prevent accidents caused by driver drowsiness [6].
* Behavioral measurements for detecting driver drowsiness mainly focus on facial features and head movements. Traditionally, machine learning algorithms were used to classify hand-crafted features such as constant eye blinking, nodding, and yawning, which were extracted using image processing techniques. However, a probabilistic model has recently been developed that tracks the driver's head, eyelid movement, and facial expressions to detect drowsiness more effectively. This approach has shown promising results in detecting driver drowsiness accurately and reliably [9].

**DATASET:**

* Driver drowsiness is a severe issue that can lead to accidents on the road. To address this problem, a system can be developed to monitor drivers' behavior and alert them when signs of drowsiness are detected. This system uses various sensors to see these signs, such as tracking eye movements, head position, and other physical cues. For instance, if the driver is yawning frequently, closing their eyes for extended periods, or making erratic head movements, the system can alert them to take a break or stop driving altogether. By using such a system, drivers can avoid accidents caused by drowsiness and remain alert and safe on the road.
* This dataset comprises high-quality images that capture instances of yawning and eye movements. Each photo is taken with the utmost care to ensure that it accurately reflects the subject's current state of alertness. This dataset can be annotated to indicate whether each instance corresponds to a tired or alert state, providing valuable insights into the relationship between these states and specific facial expressions. With this dataset, researchers can explore the nuances associated with yawning and eye movements, including frequency, duration, and intensity. This dataset is a valuable resource for anyone interested in studying human behavior and its relation to various states of consciousness.
* Deep learning models such as Convolutional Neural Network (CNN), VGG, and MobileNetV2 are commonly used to detect drowsiness in drivers. These models are trained on a dataset that typically contains many images of drivers, which are labeled as either tired or alert. The goal is to use the model to accurately predict whether a driver is in a state of drowsiness or not. In the dataset mentioned, there were 2032 images, which had been classified into four distinct classes. In this dataset, we utilized multiclass classification class mode as a categorical
* This dataset is organized into directories, where each subdirectory represents a class. Images in the dataset are in grayscale format, indicating that color information is not considered for the task. The training data includes augmentation techniques such as flipping and zooming, which enhance the model's generalization. To split the dataset into training and testing sets, specific directories (train\_data\_path and test\_data\_path) are used. If you have access to these directories, you can explore them to understand the actual content and structure of the dataset. It is essential to set up the data generator correctly since it plays a crucial role in efficiently loading and augmenting data during model training.
* This dataset comprises a diverse range of real-world scenarios, including but not limited to in-vehicle monitoring systems, surveillance footage, and controlled experimental setups. By capturing instances of individuals displaying both alertness and drowsiness, the data provides a comprehensive insight into the behavioural patterns of human subjects in various contexts.
* Drowsiness is a physiological condition when the body's arousal level is on the brink of transitioning from wakefulness to sleep. This state can be identified by various observable behaviours, such as frequent yawning, difficulty maintaining eye contact, and nodding. These signs indicate a reduced level of alertness and may lead to impaired driving ability, making it crucial for drivers to recognize and address drowsiness promptly[11].

**METHODOLOGY:**

This research aims to create a system that can detect drowsiness by implementing Convolutional Neural Networks (CNNs), focusing on two widely used architectures: VGG and MobileNetV2. The purpose is to investigate the ability of these architectures to capture and categorize facial characteristics that indicate drowsiness. The process involves gathering data, preparing it for analysis, designing the model, training it, and evaluating its effectiveness. The data instances were carefully marked and differentiated to identify the contrasting states of wakefulness and drowsiness. The criteria used to make this distinction included observations of closed or opened eyes, instances of yawning, and lack of yawning.

* **DATA GATHERING:**

The dataset in question comprises four primary classes: opened eye, closed eye, yawn, and no-yawn. With 2032 images, we utilize deep-learning models to predict drowsiness based on the symptoms above. Such predictions can potentially prevent road accidents caused by drowsiness. To detect drowsiness, we employ CNN, VGG, and MobileNetV2 models, advanced deep-learning algorithms capable of accurately identifying patterns in the input data. By training these models on the dataset, we create predictive models that can alert drivers of their drowsiness, thereby reducing the risk of accidents on the road.

* **CREATING AN ENVIRONMENT:**

Google Colab, also known as Collaboratory, is a cloud-based platform offered by Google that enables users to write and execute Python code together. This platform has gained immense popularity in deep learning and data analysis due to its seamless integration with widely used libraries such as TensorFlow and PyTorch.

* **CNN (CONVOLUTIONAL NEURAL NETWORK):**

They are addressing image recognition and object tracking challenges. CNN, a type of Artificial Neural Network (ANN), has been widely acknowledged for its superior accuracy and effectiveness in handling pixel data. Its unique ability to automatically detect essential features without human intervention makes it an ideal feature extraction and classification tool. The core structure of CNNs is composed of non-linear convolution and pooling layers that simultaneously train a vast number of filters. The first, convolution layer extracts features from an input image by performing a mathematical process using two inputs: an image matrix and a filter or kernel. The convolution layer utilizes the image pixels as input data and generates locally weighted sums for correlated regions using filters. The resulting values are referred to as feature maps. The filters are repeatedly used over the entire dataset to enhance training while reducing the number of parameters to be learned. CNNs have demonstrated remarkable performance in various complex image recognition tasks and continue to be an active area of research. Their proficiency in extracting essential features from pixel data has led to their widespread adoption in various fields, such as computer vision, autonomous vehicles, and medical image analysis.

* **Input layer:** The neural network architecture comprises multiple layers for input data processing. The first layer, the input layer, receives the initial data values as a matrix. This matrix can be one-dimensional or two-dimensional, depending on the nature of the input data.
* **Convolution layer:** The convolution layer is the second layer of a neural network, responsible for carrying out the actual computation. This layer utilizes a small integer kernel to frequently process the input data matrix mathematically depicted by the equation below.
* **Activation layer:** The activation layer limits the range of a neural network's matrix entries to enhance performance. It uses nonlinear operations like sigmoid or tanh to achieve this.
* **Pooling layer:**

After applying an activation layer, the convolution matrix may expand and become redundant, leading to inefficient use of resources and decreased performance. Employing techniques to limit matrix size and reduce redundancy is advisable.

* **VGG (VISUAL GEOMETRY GROUP):**

VGG is a deep neural network architecture proposed by the Visual Geometry Group in 2015. The architecture is known for its uniformity and simplicity, consisting mainly of convolutional layers with 3x3 filters followed by max-pooling layers. The VGG architecture has several variants, with VGG16 and VGG19 being the most popular. The primary contribution of VGG is its deep architecture, which has demonstrated that deeper networks can perform better in image classification tasks. Although newer architectures like ResNet have surpassed VGG in terms of efficiency and accuracy, VGG remains a benchmark model for developing new neural network architectures. The VGG neural network architecture is designed to learn complex and hierarchical features from input images. This is achieved through a series of convolutional layers interspersed with max-pooling layers. These layers work together to capture intricate patterns and spatial relationships in facial images that are important for detecting drowsiness. Overall, the VGG architecture is an efficient and effective tool for image classification tasks.

* **MOBILENETV2:**

The MobileNetV2 technology presents a highly effective approach for identifying drowsiness, utilizing its exceptional efficiency and compatibility for deployment on devices with limited resources. Using transfer learning and fine-tuning techniques, MobileNetV2 can be personalized to accurately detect facial expressions and signs that indicate drowsiness, facilitating the creation of highly responsive and efficient drowsiness detection systems. MobileNetV2 is an efficient neural network architecture that allows real-time inference on devices with limited computational resources. This is especially useful for applications such as in-vehicle drowsiness detection systems, where fast responses are critical. The low-latency capabilities of MobileNetV2 make it an ideal choice for such resource-constrained scenarios. Among all the three deep learning models, VGG is the best model with 84% accuracy, CNN got 76 %, and Mobilenetv2 got 77%, so VGG and CNN got the best accuracy among all the three assignments.

**EXPERIMENTAL ANALYSIS:**

Throughout the training process, a learning rate of 0.0001 was determined and applied to each frame sequence for five epochs. A batch size of 32 was also used for all frame sizes during the model training. The proposed model was implemented using Python and the TensorFlow library, which provided the necessary tools and resources.

* **Preprocessing: The initial step in pre-processing involves data collection, examination of missing values, and modifying or eliminating** certain features. This allows for the selection of relevant features in the dataset. Data cleaning is typically performed after data collection, followed by data integration. Deep learning algorithms can then be trained to determine the most optimal results for the chosen dataset. Denoising tools are often used to eliminate noise from data to achieve reliable and accurate results. This pre-processing step identifies and removes missing data fields and reduces noisy data. Finally, all unique values found within the dataset's features are extracted.
* **Implementation**: In the project, we employed the Keras library to develop a range of deep-learning models to detect drowsiness. Among these models were Convolutional Neural Networks (CNN), VGG (Visual Geometry Group), and MobilenetV2 models, which are known to deliver high accuracy and efficiency in such applications. By leveraging these advanced models, we achieved excellent results detecting when a person feels drowsy. This is crucial for ensuring safety in various settings, including driving, operating heavy machinery, etc.

1. **COMPARISION METRICS:** From the table(1), we can declare that among all the three models, VGG, CNN, and MobilenetV2 got the best accuracy for detecting drowsiness as VGG got 84%, CNN got 77%, and MobilenetV2 got 76%, all three models got approximately nearby values.

* **Accuracy:** Classification tasks often use accuracy as a critical metric. It indicates the number of correctly classified instances and the total number of cases. In event classification, TP/TN represents accurately identified positive or negative events, while FP/FN represents misclassified instances. By examining both TP/TN and FP/FN, we can gain valuable insights into a model's ability to identify and avoid misclassification.

Accuracy = [TP+TN]/[TP+TN+FP+FN]

* **Precision:** This percentage measures the accuracy of optimistic predictions about the overall number of positive cases. It indicates the proportion of positive instances that a prediction model correctly identified.

Precision = [TP]/[TP+FP]

* **Recall:** Accurately predicting positive instances is crucial in evaluating the effectiveness of a classification model. The ratio of accurately predicted positive instances to the total number of good cases in the proper positive subset is commonly used in business and academic settings. Achieving a high ratio is critical for reliable and accurate results.

Recall = [TP]/[TP+FN]

* **F1 Score:** Precision alone can lead to erroneous conclusions. It's crucial to consider recall and accuracy for a more precise outcome. F1 score balances and integrates these metrics effectively, resulting in a comprehensive outcome.

F1Score = [2\*Precision\*Recall/[Precision+ Recall]

* **Steps to predict the output**:
* The process of predicting output involves using various libraries and functions in Python. Numpy, a popular library for numerical operations, is imported alongside the load model function from the TensorFlow Keras module to load a pre-trained neural network model. The image module from Keras is imported for image processing purposes.
* First, we specify the path for the pre-trained model file ('drowiness\_new6.h5') and the test image ('Copy of 209. jpg'). The pre-trained model is loaded from the specified file path while the test image is loaded, resized to (256, 256), and converted to grayscale.
* The image is then converted to a NumPy array, and an additional dimension is added to represent the batch dimension, which is required for model input. The loaded model is then used to make predictions on the pre-processed test image.
* A threshold is applied to the predicted probabilities to convert them into binary predictions. Predictions greater than 0.5 are considered positive, while those with a probability less than or equal to 0.5 are considered negative. The binary predictions are printed and decoded into human-readable categories.
* The code assumes that the binary prediction array has a specific structure where different indices correspond to other classes, suggesting a multi-class classification task. The final prediction is based on the decoded binary prediction array and identifies the class with a predicted value 1.
* The final prediction indicates whether the model predicts the input image to represent an "Opened Eye," "Closed Eye," "Yawn," or "No Yawn" based on the assumed classification task.
* The predicted output is Binary Prediction: [[0 0 1 0]]
* According to the above output, we detected the image as a yawn.

**DISCUSSION AND FUTURE WORK:**

This research paper delves into developing a highly effective driver drowsiness detection system that harnesses the power of the CNN, VGG, and MobilenetV2 modules. Given the critical role that driver alertness plays in ensuring road safety and preventing accidents, this study sheds light on a significant issue faced by drivers worldwide. The experiment's findings showcase the system’s ability to identify crucial facial features accurately and monitor drivers' alertness levels. Overall, the research offers a comprehensive insight into the implementation and effectiveness of the drowsiness detection system.

To enhance the proposed system's effectiveness, there are two particular areas where future research and contributions could be made. Firstly, real-time implementation of the system could be explored, enabling drivers to receive timely alerts regarding their drowsiness levels. This could be achieved by using a portable device or an in-car camera. By implementing the system in real-time, drivers would have a better chance of responding to the alerts and taking the necessary measures to avoid accidents. Secondly, personalization of the system could also be considered. This would involve customizing the system based on the driver's age, gender, and driving history. By doing so, the system would have a higher accuracy in detecting drowsiness, as it would be tailored to the individual driver's characteristics and habits. This could be achieved through further research and data analysis related to drowsiness and driving behavior, which could then be used to develop a personalized algorithm for the system [3]. We have developed a driver drowsiness detection system that accurately monitors a driver's level of drowsiness. Our model achieved an impressive accuracy rate of 84% during training, making it a reliable solution for detecting driver drowsiness and preventing accidents. While the system has shown promising results, it may not perform as well with larger datasets in real-time applications. Despite this limitation, our proposed model offers a viable solution for enhancing road safety.[4]

**REFERENCES:**

1. Satapathy, S. K., Mori, H., Naik, K., & Rajput, N. S. (2023, April). Automated Drowsiness Detection for Driver Safety: A Deep Learning-based Approach. In 2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES) (pp. 774-779). IEEE.
2. Ganguly, B., Dey, D., & Munshi, S. (2022, February). An integrated system for drivers’ drowsiness detection using deep learning frameworks. In 2022 IEEE VLSI Device Circuit and System (VLSI DCS) (pp. 55-59). IEEE.
3. Ojha, D., Pawar, A., Kasliwal, G., Raut, R., & Devkar, A. (2023, May). Driver Drowsiness Detection Using Deep Learning. In 2023 4th International Conference for Emerging Technology (INCET) (pp. 1-4). IEEE
4. Ch, S. P., Guduru, S., Ronagala, V., Kuresan, H., & Dhanalakshmi, S. (2023, January). Automatic System for Driver Drowsiness Detection System using Deep Learning. In 2023 International Conference for Advancement in Technology (ICONAT) (pp. 1-4). IEEE.
5. Ahuja, H., Saurav, S., Srivastava, S., & Shekhar, C. (2020, December). Driver drowsiness detection using knowledge distillation technique for real-time scenarios. In 2020 IEEE 17th India Council International Conference (INDICON) (pp. 1-5). IEEE
6. Nandhini, P. S., Kuppuswami, S., Malliga, S., Srinath, P., & Veeramanikandan, P. (2022, March). Driver drowsiness detection using deep learning. In 2022, the 6th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 1031-1036). IEEE.
7. Vijaypriya, V., & Uma, M. (2023, April). Comparative Analysis for Detecting Drowsiness Using Deep Learning Approach. In 2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT) (pp. 1-4). IEEE
8. Tümen, V., Yıldırım, Ö., & Ergen, B. (2018, April). Detection of driver drowsiness in driving environment using deep learning methods. In 2018 Electric Electronics, Computer Science, biomedical Engineerings' Meeting (EBBT) (pp. 1-5). IEEE.
9. Aydemir, G., Kurnaz, O., Bekiryazıcı, T., Avcı, A., & Kocakulak, M. (2021, November). Driver Drowsiness Detection using MobileNets and Long Short-term Memory. In 2021, the 13th International Conference on Electrical and Electronics Engineering (ELECO) (pp. 220-223). IEEE
10. Suguna, R., Priya, M. S., Hema, S., Aravind, K. D., Gowtham, A. M., & Vignesh, S. (2023, August). Deep Learning Based Driver Drowsiness Monitoring and Alert System. In 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT) (pp. 241-246). IEEE.
11. Pawar, R., Wamburkar, S., Deshmukh, R., & Awalkar, N. (2021, October). Driver drowsiness detection using deep learning. In 2021 2nd Global Conference for Advancement in Technology (GCAT) (pp. 1-7). IEEE.

**APPENDIX 1:**

**Screenshot 1. userID**

**A screenshot of a computer

Description automatically generated**

**Screenshot 2. CNN model epochs screenshot**

A screenshot of a computer

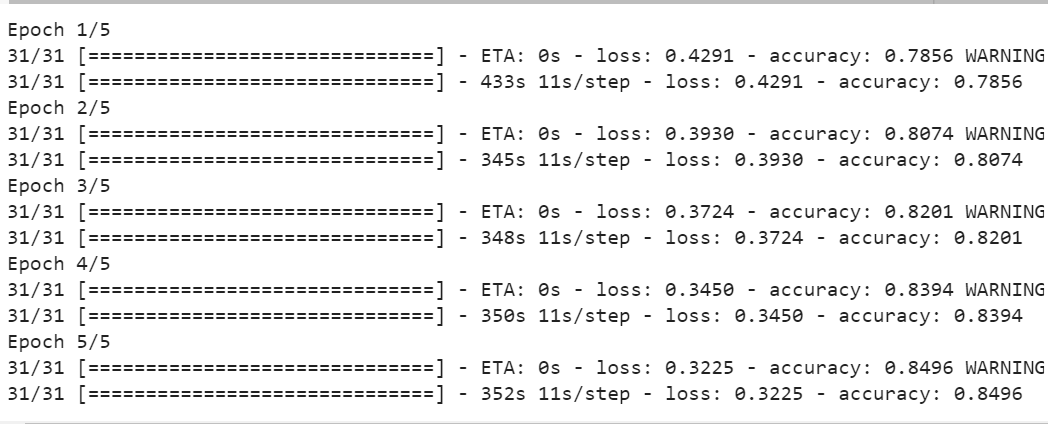
Description automatically generated

A graph with a line

Description automatically generated

**Figure 1. Accuracy Graph of CNN model**

**Screenshot 3.VGG model epochs**



A graph with a line

Description automatically generated

**Figure 2. Accuracy Graph of VGG model**

**Screenshot 4.MobilenetV2 epochs**

**A number and numbers on a white background

Description automatically generated**

**A graph with a line

Description automatically generated**

**Figure 3. Accuracy Graph for MobilenetV2 Model**

**Figure 4. Image to predict output:**

**A close up of a person's eye

Description automatically generated**

**Steps to predict output:**

* The process of predicting output involves the use of various libraries and functions in Python. Numpy, a popular library for numerical operations, is imported alongside the load model function from the TensorFlow Keras module to load a pre-trained neural network model. The image module from Keras is imported for image processing purposes.
* To start with, we specify the file paths for both the pre-trained model file ('drowiness') and the test image ('Copy of 01. jpg'). The pre-trained model is loaded from the specified file path while the test image is loaded, resized to (256, 256), and converted to grayscale.
* The converted image is then converted to a NumPy array and an additional dimension is added to represent the batch dimension, which is required for model input. The loaded model is then used to make predictions on the pre-processed test image.
* A threshold is applied to the predicted probabilities to convert them into binary predictions. Predictions with a probability greater than 0.5 are considered positive, while those with a probability less than or equal to 0.5 are considered negative. The binary predictions are printed and decoded into human-readable categories.
* The code assumes that the binary prediction array has a specific structure where different indices correspond to other classes, suggesting a multi-class classification task. The final prediction is based on the decoded binary prediction array and identifies the course with a predicted value 1.
* The final prediction indicates whether the model predicts the input image to represent an "Opened Eye," "Closed Eye," "Yawn," or "No Yawn" based on the assumed classification is Binary Prediction: [[1 0 0 0]]
* We detected the image as the opened eye according to the above output.

**Image:**

**STEP 1.**

**A screenshot of a computer

Description automatically generated**

**STEP 2.**

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **Model name** | **Accuracy** |
| VGG(Visual geometry group) | **84%** |
| CNN(Convolutional Network) | **77%** |
| MobilenetV2 | **76%** |

**RESULT TABLE 1:**

**APPENDIX 2:**

#Importing all the libraries

import numpy as np

import matplotlib.pyplot as plt

plt.style.use('dark\_background')

import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten, Conv2D, MaxPooling2D, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf

from tensorflow.keras.applications.inception\_v3 import preprocess\_input

import numpy as np

from tensorflow.keras.preprocessing import image

from tensorflow.keras.models import load\_model

import os

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import classification\_report

from matplotlib.pyplot import imshow

from tensorflow.keras.preprocessing import image

from PIL import Image

import socket

print(socket.gethostname())

pip install --upgrade tensorflow

Function to plot image

def plot\_imges(directory, top=10):

    all\_item\_dirs = os.listdir(directory)

    item\_files = [os.path.join(directory, file) for file in all\_item\_dirs][:5]

    plt.figure(figsize=(20, 20))

    for i, img\_path in enumerate(item\_files):

        plt.subplot(10, 10, i+1)

        img = plt.imread(img\_path)

        plt.tight\_layout()

        plt.imshow(img, cmap='gray')

data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/train'

directories = ['/Closed', '/Open', '/no\_yawn', '/yawn']

**plotting sample images**

for j in directories:

    plot\_imges(data\_path+j)

train\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/train'

test\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/test'

batch\_size = 128

# Set data paths and other parameters

train\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/train'

test\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/test'

batch\_size = 64

sequence\_lengthh = 10

num\_epochs = 10

num\_class = 4

# Create data generators

train\_datagen = ImageDataGenerator(

    horizontal\_flip=True,

    rescale=1./255,

    zoom\_range=0.2,

    validation\_split=0.1

)

train\_set = train\_datagen.flow\_from\_directory(

    train\_data\_path,

    target\_size=(256, 256),

    batch\_size=batch\_size,

    color\_mode='grayscale',

    class\_mode='categorical'

)

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_set = test\_datagen.flow\_from\_directory(

    test\_data\_path,

    target\_size=(256, 256),

    batch\_size=batch\_size,

    color\_mode='grayscale',

    class\_mode='categorical'

)

CNN model

# Build the model

model = Sequential()

model.add(Conv2D(32, (3, 3), padding="same", activation="relu", input\_shape=(256, 256, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(256, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dropout(0.5))

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

model.add(Dense(4, activation="softmax"))

model.summary()

model.compile(loss="categorical\_crossentropy", metrics=["accuracy"], optimizer="adam")

# Set up model checkpoint

model\_path = "/content/drive/MyDrive/drowsiness/dataset\_new/train"

checkpoint = ModelCheckpoint(model\_path, monitorr='val\_accuracy', verbose=1, save\_best\_only=True, mode='max')

callback\_list = [checkpoint]

# Plot training history

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

model.save("drowiness\_new6.h5")

VGG Model

# Build VGG-like model

model = Sequential()

model.add(Conv2D(32, (3, 3), padding="same", activation="relu", input\_shape=(256, 256, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(256, (3, 3), padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dropout(0.5))

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

model.add(Dense(num\_class, activation="softmax"))

# Print the model summary

model.summary()

# Compile the model

model.compile(loss="categorical\_crossentropy", metrics=["accuracy"], optimizer="adam")

# Create data generators

train\_datagen = ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1./255)

train\_set = train\_datagen.flow\_from\_directory(train\_data\_path, target\_size=(256, 256), batch\_size=batch\_size, class\_mode='categorical', shuffle=True)

test\_set = test\_datagen.flow\_from\_directory(test\_data\_path, target\_size=(256, 256), batch\_size=batch\_size, class\_mode='categorical', shuffle=False)

# Set up callbacks, modelcheckpoint

checkpoint = ModelCheckpoint("vgg\_model.h5", monitorr="val\_accuracy", save\_best\_only=True, mode="max")

# Set up model checkpoint

model\_path = "/content/drive/MyDrive/drowsiness/dataset\_new/train"

checkpoint = ModelCheckpoint(model\_path, monitor='val\_accuracy', verbose=1, save\_best\_only=True, mode='max')

callbacks\_list = [checkpoint]

# Set data paths and other parameters

train\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/train'

test\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/test'

batch\_size = 64

sequence\_lengthh = 10

num\_epochs = 5

num\_class = 4

# Create data generators

train\_datagen = ImageDataGenerator(

    horizontal\_flip=True,

    rescale=1./255,

    zoom\_range=0.2,

    validation\_split=0.1

)

train\_set = train\_datagen.flow\_from\_directory(

    train\_data\_path,

    target\_size=(256, 256),

    batch\_size=batch\_size,

    color\_mode='grayscale',

    class\_mode='categorical'

)

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_set = test\_datagen.flow\_from\_directory(

    test\_data\_path,

    target\_size=(256, 256),

    batch\_size=batch\_size,

    color\_mode='grayscale',

    class\_mode='categorical'

)

#Fitting the model

num\_epochs = 5

training\_steps=train\_set.n//train\_set.batch\_size

validation\_steps =test\_set.n//test\_set.batch\_size

history = model.fit(train\_set, epochs = num\_epochs, steps\_per\_epoch=training\_steps, validation\_data=test\_set, validation\_steps=validation\_steps, callbacks = callbacks\_list)

# Plot training and validation accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# MobilenetV2 Model

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

# Create MobileNetV2 base model

base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(256, 256, 1))

# Freeze the layers of the base model

for layer in base\_model.layers:

    layer.trainable = False

# Adding custom layers for classification task

x = GlobalAveragePooling2D()(base\_model.output)

x = Dense(128, activation='relu')(x)

output = Dense(num\_class, activation='softmax')(x)

# Create the final model

model = Model(inputs=base\_model.input, outputs=output)

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Display the model summary

model.summary()

# Set data paths and other parameters

train\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/train'

test\_data\_path = '/content/drive/MyDrive/drowsiness/dataset\_new/test'

batch\_size = 64

sequence\_length = 10

num\_epochs = 5

num\_class = 4

# Create data generators

train\_datagen = ImageDataGenerator(

    horizontal\_flip=True,

    rescale=1./255,

    zoom\_range=0.2,

    validation\_split=0.1

)

train\_set = train\_datagen.flow\_from\_directory(

    train\_data\_path,

    target\_size=(256, 256),

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='training'

)

val\_set = train\_datagen.flow\_from\_directory(

    train\_data\_path,

    target\_size=(256, 256),

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='validation'  # Use the validation split

)

# Fit the model to the data

history = model.fit(

    train\_set,

    epochs=num\_epochs,

    validation\_data=val\_set

)

# Plot training and validation accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

OUTPUT:

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

# Define the paths

model\_path = ' /content/drive/MyDrive/drowsiness’

test\_image\_path = r' /content/drive/MyDrive/drowsiness/Copy of \_4.jpg’

# Load and preprocess the test image

img = image.load\_img(test\_image\_path, target\_size=(256, 256), color\_mode="grayscale")

img\_array = image.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0)  # Add batch dimension

# Make predictions

predictions = model.predict(img\_array)

# Assuming binary classification, you can threshold the predictions

binary\_prediction = (predictions > 0.5).astype(int)

# Print or use the binary prediction as needed

print("Binary Prediction:", binary\_prediction)

# Decode the binary prediction into human-readable categories

if binary\_prediction[0][0] == 1:

    print("Predicted: Opened Eye")

elif binary\_prediction[0][1] == 1:

    print("Predicted: Closed Eye")

elif binary\_prediction[0][2] == 1:

    print("Predicted: Yawn")

else:

    print("Predicted: No Yawn")