PRACTICAL TRAINING I REPORT

On

DROWSINESS DETECTION SYSTEMS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD

OF

DEGREE OF BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING-ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Gurugram University, Gurugram

Under the Guidance of
GOVIND GUPTA
Designation

***Submitted By*:**
Name: BHAVISHYA CHATURVEDI
Roll No. 25232

June-July 2024

Dronacharya College of Engineering, Gurugram

DRONACHARYA College of Engineering

STUDENT DECLARATION

I hereby declare that the Practical Training Report entitled " **DROWSINESS DETECTION SYSTEM" is** an authentic record of my own work as requirements of 5th semester during the period from 11-06-2024 to 23-07-2024 for the award of degree of Btech. (Computer Science & Engineering-Artificial Intelligence & Machine Learning), Dronacharya College of Engineering.

Bhavishya Chaturvedi

Bhavishya Chaturvedi
25232

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Signatures

Examined by:

Head of Department

(Dr. Ritu Pahwa)

DRONACHARYA
College of Engineering

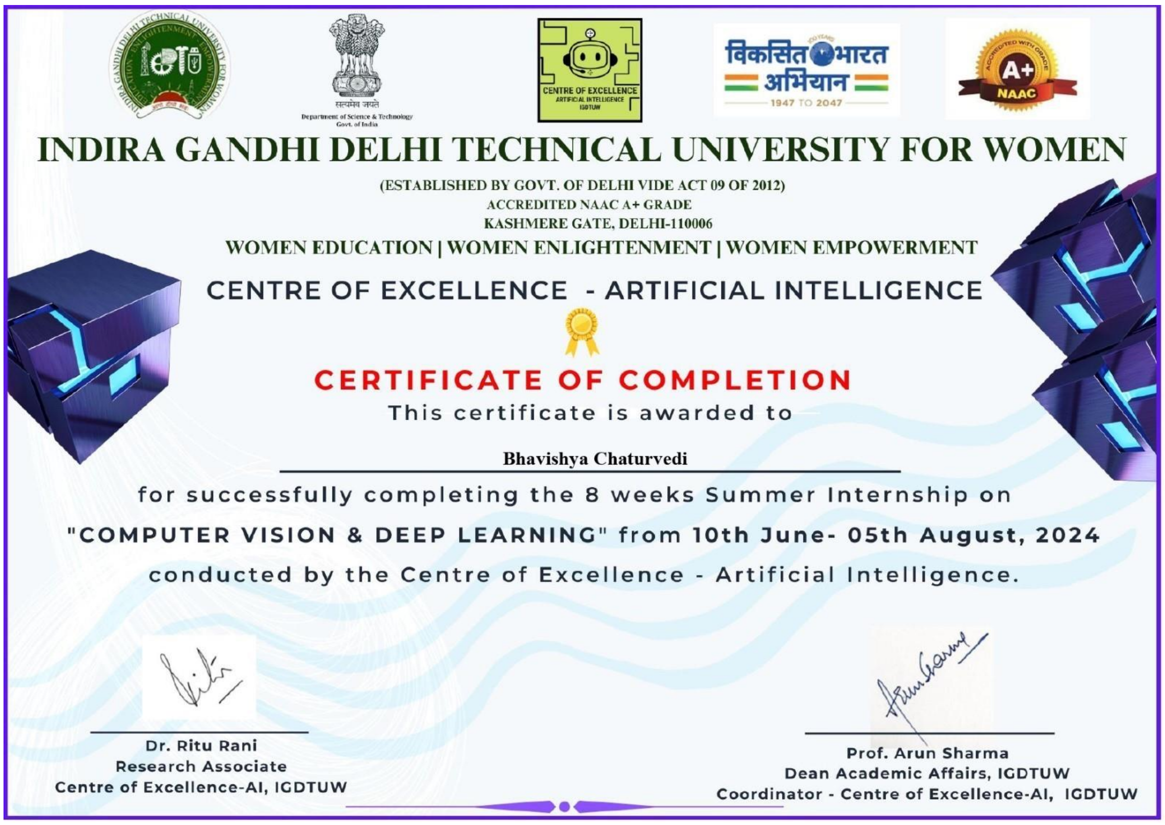
ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my training guide " Govind Gupta", “Lab Instructor, and any other” for giving me the opportunity to work on this topic. It would never be possible for us to take this training to this level without his innovative ideas and his relentless support and encouragement.

I would also like to express my sincere gratitude to my Head of Department Ritu Pahwa for constant support and motivation. It would never be possible for us to take this training to this level without her innovative ideas and her relent less support and encouragement.

Bhavishya Chaturvedi
25232

Internship Certificate



INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

(ESTABLISHED BY GOVT. OF DELHI VIDE ACT 09 OF 2012)
ACCREDITED NAAC A+ GRADE
KASHMERE GATE, DELHI-110006

WOMEN EDUCATION | WOMEN ENLIGHTENMENT | WOMEN EMPOWERMENT

CENTRE OF EXCELLENCE - ARTIFICIAL INTELLIGENCE

CERTIFICATE OF COMPLETION

This certificate is awarded to

Bhavishya Chaturvedi

for successfully completing the 8 weeks Summer Internship on "COMPUTER VISION & DEEP LEARNING" from 10th June- 05th August, 2024 conducted by the Centre of Excellence - Artificial Intelligence.

Dr. Ritu Rani
Research Associate
Centre of Excellence-AI, IGDTUW

Prof. Arun Sharma
Dean Academic Affairs, IGDTUW
Coordinator - Centre of Excellence-AI, IGDTUW

Contents Page

Chapter 1

INTRODUCTION

Drowsiness detection systems play a crucial role in preventing accidents, especially in environments where alertness is paramount. Traditional methods often rely on simple sensors or behavioural observations, which can be unreliable due to individual variations, environmental conditions, and the inherent subjectivity of human judgment. To address these limitations, this research proposes a more advanced drowsiness detection system utilizing deep learning and computer vision techniques.

By leveraging the power of convolutional neural networks (CNNs) and OpenCV, the system will analyse video streams, focusing on facial expressions, eye movements, and head poses to identify signs of fatigue. This approach offers several advantages over traditional methods, including the ability to handle complex visual patterns, adapt to different lighting conditions, and provide more objective and consistent results.

The primary objectives of this research are to develop a highly accurate, robust, efficient, and user-friendly drowsiness detection system. The system should be capable of operating in real-world environments, even under challenging conditions, and provide reliable detection of drowsiness in real-time. Additionally, it should be resilient to variations in facial features, lighting, and background noise, and process video streams efficiently to enable real-time applications. Finally, the system should provide visual alerts and notifications that are easy to understand and act upon.

To achieve these objectives, the proposed system will follow a methodology that involves data collection, preprocessing, feature extraction, classification, and integration with OpenCV. By gathering a diverse dataset of facial images and videos depicting both alert and drowsy individuals, the system will be trained to recognize the subtle cues associated with fatigue. Through preprocessing techniques, the data will be prepared for analysis, ensuring consistency and improving model performance.

CNNs will be employed to extract relevant facial features, such as eye closure, head tilt, and facial muscle activity. These features will then be used to train a deep learning model that can accurately classify facial images and videos into "alert" or "drowsy" categories. The trained model will be integrated with OpenCV to enable real-time video processing and visual alerts, providing users with timely notifications of potential drowsiness.

The expected outcomes of this research include a highly accurate drowsiness detection system capable of operating in real-world environments, improved safety and efficiency in various applications, and contributions to advancements in computer vision and deep learning for human behaviour analysis. By addressing the limitations of traditional methods and leveraging the power of modern deep learning techniques, this research aims to develop a more reliable and effective tool for preventing accidents and ensuring safety in environments where alertness is critical

Chapter 2

OBJECTIVE

Internship are generally thought of to be reserved for college students looking to gain experience in a particular field. However, a wide array of people can benefit from Training Internships to receive real world experience and develop their skills. Utilizing internships is a great way to build your resume and develop skills that can be emphasized in your resume for future jobs.

Internships benefit both the student and the employee. On-the-job learning reinforces what you see in the classroom and teaches invaluable skills like time management, communication, working with others, problem-solving, and, most importantly, the willingness to learn. For employers, you can build relationships and prepare future employees.

Most college graduates will have to overcome a barrier to entry due to an experience requirement — something college students do not always get with their education. An internship provides you with firsthand experience, professional opportunities and personal growth.

It will also make you more competitive when applying for jobs. As an intern, you'll gain relevant skills to showcase on your resume. After a successful internship, it's common to receive a letter of recommendation or a potential job offer.

An internship expands your professional network. Interns often report to mid- to senior-level positions. These mentors can help guide you and provide helpful advice for your career path. Plus, you will meet others within the company who have diverse work experience and professional connections.

Chapter 3

3.1 Computer Vision with OpenCV

OpenCV (Open Source Computer Vision Library) is a widely-used library for image processing and computer vision tasks. It provides tools for image manipulation, feature extraction, and object detection, making it a valuable resource for building drowsiness detection systems.

Bradski and Kaehler (2008) describe the capabilities of OpenCV in their book "Learning OpenCV: Computer Vision with the OpenCV Library." The library offers various algorithms for image filtering, edge detection, and color space conversion, which are essential for preprocessing images before applying machine learning models.

3.2 Machine Learning and CNNs

Convolutional Neural Networks (CNNs) have become the standard for image classification tasks due to their ability to learn hierarchical features from raw image data. CNNs are particularly effective for detecting patterns in images, such as the presence of closed or semi-closed eyes.

LeCun, Bengio, and Hinton (2015) provide a comprehensive overview of deep learning techniques in their seminal paper "Deep Learning," published in *Nature*. CNNs are designed to automatically learn and extract features from images, which makes them suitable for tasks like face detection.

Keras, a high-level API for building deep learning models, simplifies the process of creating and training CNNs. Chollet (2015) introduced Keras in his documentation, emphasizing its ease of use and flexibility. Keras interfaces with various backend engines, including TensorFlow, which provides the computational power required for training complex models.

3.3 Drowsiness Detection Models

Traditional drowsiness detection methods often rely on manual feature extraction and classification algorithms. Techniques include analysing color patterns, textures, and shapes in images to identify facial features.

Hsu and Wang (2015) explored hybrid feature selection methods for drowsiness detection in their paper "drowsiness detection using hybrid feature selection," published in the *Journal of Vehicle safety*. Their approach combines various image features and classification algorithms to improve detection accuracy.

Recent research has shifted towards using deep learning models, particularly CNNs, for automatic face detection. Chen et al. (2020) presented a CNN-based approach for drowsiness detection in their paper "Face detection with convolutional neural networks," presented at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Their method leverages large-scale datasets and advanced CNN architectures to achieve high accuracy.

3.4 Implementation in Google Colab

Google Colab provides a cloud-based environment for running Python code, including machine learning models. It offers free access to GPUs, which are essential for training deep learning models efficiently.

Bisong (2019) highlights the benefits of Google Colab in his book "Building Machine Learning and Deep Learning Models on Google Cloud Platform." Colab allows users to develop and test machine learning models without requiring local computational resources, making it an ideal platform for prototyping drowsiness detection systems.

3.5 Challenges and Future Directions

Despite advancements, real-time detection of drowsiness remains challenging due to varying environmental conditions and image quality. Karami and Moeslund (2021) discuss these challenges in their paper "Real-time facial detection in video streams using convolutional neural networks," published in *Computers*. They address the need for robust models that can handle diverse scenarios and maintain high accuracy.

Another challenge is the availability of high-quality datasets for training and evaluating models. Liu et al. (2022) reviewed datasets for facial feature detection in their paper "A comprehensive review of datasets for drowsiness detection," published in *Data*. They note that while open-source datasets are growing, there is still a need for more comprehensive and diverse datasets to improve model performance.

Certainly! Here’s a detailed description of the software components involved in the drowsiness detection system, explaining their roles and how they contribute to the system's overall functionality.

Chapter 4

SOFTWARE REQUIREMENTS SPECIFICATIONS

4.1 Python

• Overview

Python is a versatile, high-level programming language known for its simplicity and readability. Developed by Guido van Rossum, Python's syntax and dynamic typing make it a popular choice for both beginners and experienced programmers. It supports multiple programming paradigms, including object-oriented, imperative, and functional programming.

• Role in Drowsiness Detection:

Python is ideal for rapid application development and scripting, which is essential for building and prototyping a drowsiness detection system. Its extensive standard library provides modules for various tasks, including data manipulation, file handling, and network communication, which are crucial for developing a comprehensive detection system.

• Advantages:

Ease of Learning and Use: Python's clear syntax and dynamic typing facilitate easy coding and debugging.

Extensive Libraries: Python's rich ecosystem of libraries and frameworks, including those for data science and machine learning, enhances development efficiency.
Cross-Platform Compatibility: Python runs on various platforms, making it suitable for developing cross-platform applications.

Libraries and Modules:

- NumPy and Pandas: For data manipulation and analysis.
- OpenCV: For image processing tasks.
- Matplotlib and Seaborn: For data visualization.

4.2 Deep Learning

- Overview:

Deep learning is a subset of machine learning and artificial intelligence, focusing on neural networks with multiple layers (deep neural networks). It excels at extracting and transforming features through layers of nonlinear processing units, making it suitable for complex tasks such as image and speech recognition.

- Role in Drowsiness Detection:

Deep learning models, particularly convolutional neural networks (CNNs), are used to analyse and classify images. These models learn hierarchical representations of features from raw image data, enabling accurate detection of drowsiness patterns.

- Key Concepts:

Neural Networks: Composed of layers of interconnected nodes (neurons), each layer transforms input data and passes it to the next layer.
Feature Extraction:** Deep learning algorithms automatically extract relevant features from images, reducing the need for manual feature engineering.

- Advantages:

High Accuracy: Capable of achieving state-of-the-art performance in image classification and object detection.
Automatic Feature Learning: Learns relevant features directly from data, improving model performance.

- Applications:

Computer Vision: Used for image recognition and classification tasks in the drowsiness detection system.

4.3 OpenCV Python

- Overview:

OpenCV (Open Source Computer Vision Library) is an open-source library aimed at real-time computer vision. The Python bindings for OpenCV allow developers to use its functionalities in Python applications. OpenCV provides tools for image and video processing, including operations like filtering, edge detection, and feature extraction.

- Role in Drowsiness Detection:

OpenCV is used for preprocessing images before they are fed into the deep learning model. It handles tasks such as resizing, normalization, and enhancement, which prepare the images for more accurate analysis by the CNN.

- Key Features:

Image Processing: Functions for image resizing, smoothing, and feature extraction.
Video Analysis: Capabilities for capturing and processing video streams in real-time.

- Advantages:

Efficiency: Optimized for performance with a focus on real-time applications.
Extensive Functionality: Provides a wide range of tools for image and video manipulation.

- Common Functions:

cv2.resize(): Resizes images to the required dimensions.
cv2.normalize(): Scales pixel values to a specific range.

4.4 Keras

- Overview:

Keras is an open-source neural network library written in Python. It serves as a high-level interface for building and training deep learning models and is now fully integrated with TensorFlow. Keras simplifies the creation of neural networks with its user-friendly API and modular design.

- Role in Drowsiness Detection:

  Keras is used for designing, training, and evaluating the deep learning models that perform drowsiness detection. It provides a range of tools and pre-built layers that facilitate rapid model development and experimentation.

- Key Features:

  Modularity: Offers a modular approach to building neural networks, allowing for easy experimentation with different architectures.

  -Extensive Layers and Tools: Includes various pre-built layers, activation functions, optimizers, and loss functions.

- Advantages:

  Ease of Use: Simplifies the process of building complex neural networks with intuitive APIs.

  Integration with TensorFlow: Provides compatibility with TensorFlow's extensive functionalities and optimizations.

- Common Components:

  Layers: Dense, convolutional, pooling, dropout, etc.

  Optimizers: Adam, SGD, RMSprop, etc.

  Loss Functions: Categorical cross-entropy, binary cross-entropy, etc.

4.5 TensorFlow

- Overview:

TensorFlow is an open-source machine learning library developed by Google. It supports both training and inference of deep neural networks and is designed for high-performance numerical computations using dataflow graphs.

- Role in Drowsiness Detection:

TensorFlow serves as the backend for Keras, providing the computational power and efficiency required for training and running deep learning models. It supports distributed computing across CPUs, GPUs, and TPUs, making it suitable for handling large-scale data and complex models.

- Key Features:

Dataflow Graphs: Represents computations as dataflow graphs, where nodes are operations and edges are data.
Flexible Architecture: Allows deployment across various platforms, including desktops, mobile devices, and edge devices.

- Advantages:

Scalability: Capable of scaling from a single device to distributed systems.
Performance: Optimized for both training and inference on CPUs and GPUs.

- Common Components:

Tensors: Multi-dimensional arrays used as the fundamental data structure in TensorFlow.
TensorFlow Serving: A flexible, high-performance serving system for machine learning models.

Chapter 5

APPROACH

Approach to Building a Drowsiness Detection Model
The approach to developing a drowsiness detection model involves several key steps, combining image processing techniques with advanced deep learning methods. The primary goal is to create a system that can accurately identify human faces and expressions in real-time using a convolutional neural network (CNN) integrated with OpenCV for preprocessing and Google Colab for computational efficiency.

5.1 Data Collection and Preprocessing

The first step in building an effective drowsiness detection model is collecting and preprocessing a comprehensive dataset. A diverse dataset is essential to ensure that the model can generalize well across different scenarios. This dataset typically includes images and videos of sleepy, drowsy, and awake (normal) conditions. High-quality datasets are crucial for training robust models, and publicly available datasets such as the "Facial Features Dataset" can be used.

Preprocessing involves several tasks, including resizing images to a consistent dimension, normalizing pixel values, and augmenting the data to improve model robustness. Data augmentation techniques, such as rotation, flipping, and color adjustments, help create a more varied training dataset, which is essential for handling real-world variations in lighting, shadows, and feature sharpness intensity.

OpenCV plays a crucial role in preprocessing by providing tools for image manipulation and enhancement. Techniques such as color space conversion and edge detection can be used to highlight features relevant to face, which helps in improving the performance of the CNN model.

5.2 Model Architecture

The core of the drowsiness detection system is a Convolutional Neural Network (CNN). CNNs are well-suited for image classification tasks due to their ability to automatically learn hierarchical features from raw image data. The architecture of the CNN typically consists of several convolutional layers, pooling layers, and fully connected layers.

Convolutional Layers: These layers apply convolutional filters to the input images to detect local patterns such as edges, textures, and shapes. The filters are learned during the training process, allowing the network to identify relevant features for distinguishing between sleepy, drowsy , and normal conditions.

Pooling Layers: Pooling operations, such as max pooling, reduce the spatial dimensions of the feature maps while retaining essential information. This helps in reducing the computational load and improving the model's ability to generalize.

Fully Connected Layers: After several convolutional and pooling layers, the feature maps are flattened and passed through fully connected layers to produce the final classification output. These layers combine the extracted features to make predictions about the presence of sleepy or closed eyes.

5.3 Training and Validation

Training the CNN involves feeding it the preprocessed images and corresponding labels (sleepy, drowsy, or awake/normal). The model learns to classify images based on these labels by minimizing a loss function, such as categorical cross-entropy, using optimization algorithms like Adam or SGD (Stochastic Gradient Descent). The loss function measures the difference between the predicted and actual labels, and the optimization algorithm adjusts the model's parameters to minimize this difference.

To evaluate the model's performance, a validation set is used during training. This set consists of images that are not included in the training data but are used to assess how well the model generalizes to unseen data. Metrics such as accuracy, precision, recall, and F1-score are used to measure the model's effectiveness in detecting drowsiness.

5.4 Deployment and Real-Time Detection

Once trained, the model can be deployed for real-time detection. Google Colab provides a cloud-based environment that allows for efficient model training and testing. For real-time applications, the model can be integrated with video streams or camera feeds using OpenCV to continuously analyse incoming frames. The model's predictions are used to trigger alerts or actions based on the detection of sleep or drowsiness.

5.5 Challenges and Improvements

Despite the advances, challenges such as varying environmental conditions, image quality, and computational constraints remain. Continual improvements can be made by incorporating more diverse datasets, optimizing model architectures, and utilizing transfer learning techniques to leverage pre-trained models for enhanced performance.

Certainly! Here's a detailed description of the architecture for a drowsiness detection model specifically tailored for image detection, using Convolutional Neural Networks (CNNs) with OpenCV for preprocessing and Keras for model building.

Chapter 6

Detailed Architecture of model

6.1 Introduction

The goal of the drowsiness detection model is to accurately classify images into categories such as sleepy, drowsy, or awake (normal). This involves using a Convolutional Neural Network (CNN) to learn and identify patterns associated with these categories. The architecture is designed to preprocess images, extract features using the CNN, and perform classification.

6.2 Data Collection and Preprocessing

- Data Collection:

A high-quality dataset is essential for training a robust model. The dataset should include a variety of images representing different scenarios, such as:
- Sleepy: Consisting of closed eyes for a certain time.
- Drowsy: Different variations of semi closed eyes.
- Awake: Images without drowsiness or sleeping eyes, covering various backgrounds and conditions.

Publicly available datasets or custom-collected images can be used. Ensure the dataset is annotated with labels for each category (sleepy, drowsy, normal).

- Preprocessing:

Before feeding images into the CNN, several preprocessing steps are applied:

- Resizing: Images are resized to a uniform dimension (e.g., 224x224 pixels). Consistent image size is crucial for input into the CNN.

- Normalization: Pixel values are scaled to a range between 0 and 1 by dividing by 255. This normalization helps stabilize and accelerate training.

- Color Space Conversion (Optional): Depending on the model's design, images may be converted to grayscale or other color spaces (e.g., HSV) to enhance specific features relevant to drowsiness detection.

- Data Augmentation: To improve model robustness, data augmentation techniques are applied. These may include random rotations, flipping, cropping, and brightness adjustments. Data augmentation helps the model generalize better to unseen images.

OpenCV is used for these preprocessing tasks, providing functions for resizing, normalization, and augmentation.

6.3 CNN Architecture

The CNN architecture is designed to extract hierarchical features from images and classify them. The typical architecture includes several convolutional layers, pooling layers, and fully connected layers:

- Convolutional Layers:

- Initial Convolutional Layer: Applies a set of convolutional filters (e.g., 32 filters with a 3x3 kernel) to detect basic features like edges and textures. This layer captures low-level patterns.

- Intermediate Convolutional Layers: Subsequent layers use more filters (e.g., 64 or 128) and larger kernel sizes if needed, to detect more complex features. These layers build on the features detected by previous layers.

- Activation Function: The ReLU (Rectified Linear Unit) activation function is applied after each convolutional operation. ReLU introduces non-linearity, enabling the network to learn complex patterns.

- Pooling Layers:

- Max Pooling: Max pooling layers (e.g., 2x2 pooling) are used to downsample the feature maps, reducing their spatial dimensions while retaining the most important information. This reduces computational complexity and helps the model become invariant to small translations.

- Fully Connected Layers:

- Flattening: After several convolutional and pooling layers, the feature maps are flattened into a 1D vector. This vector contains the extracted features that will be used for classification.

- Dense Layers: Fully connected (dense) layers are added to the network. These layers combine the extracted features to make predictions. A common configuration is to use one or two dense layers with a moderate number of units.

- Output Layer: The final dense layer uses a softmax activation function to output class probabilities. For a drowsiness detection model, the output layer typically has three units corresponding to the classes: sleepy, drowsiness, and normal.

- Dropout :

- Dropout Layers: Dropout is applied to some of the dense layers to prevent overfitting. During training, dropout randomly sets a fraction of the input units to zero, which helps regularize the model and improve generalization.

6.4 Training and Validation

- Loss Function:

- Categorical Cross-Entropy: For multi-class classification, categorical cross-entropy is used as the loss function. It measures the difference between the predicted probabilities and the true labels.

- Optimizer:

- Adam Optimizer: The Adam optimizer is often used for training CNNs. It adapts the learning rate and incorporates momentum to accelerate convergence.

- Evaluation Metrics:

- Accuracy, Precision, Recall, and F1-Score: These metrics are used to evaluate the model's performance. Accuracy measures overall correctness, while precision, recall, and F1-score provide insights into how well the model detects each class (sleepy, drowsy, normal).

- Validation Set:

- Validation Data: A portion of the dataset is set aside as a validation set. This set is used to monitor the model's performance during training and adjust hyperparameters to avoid overfitting.

6.5 Implementation in Keras

The model is implemented using the Keras API, which provides a high-level interface for building and training CNNs. The architecture is defined using Keras' Sequential API or Functional API, depending on the complexity of the network.

- Defining the Model:

- Sequential API: For simpler architectures, the Sequential API is used to stack layers in a linear fashion.

- Functional API: For more complex architectures or custom configurations, the Functional API allows for greater flexibility in defining the network.

- Compiling the Model:

- Compile Configuration: The model is compiled with the categorical cross-entropy loss function, Adam optimizer, and relevant metrics.

- Training the Model:

- Fit Method: The model is trained using the `fit` method, which takes in the training data, validation data, batch size, and number of epochs. Data augmentation is applied during training to enhance model robustness.

- Evaluating the Model:

Evaluate Method: After training, the model's performance is evaluated on the test set using the `evaluate` method, providing insights into how well the model generalizes to unseen data.

Chapter 7

MODEL

7.1 Dataset overview

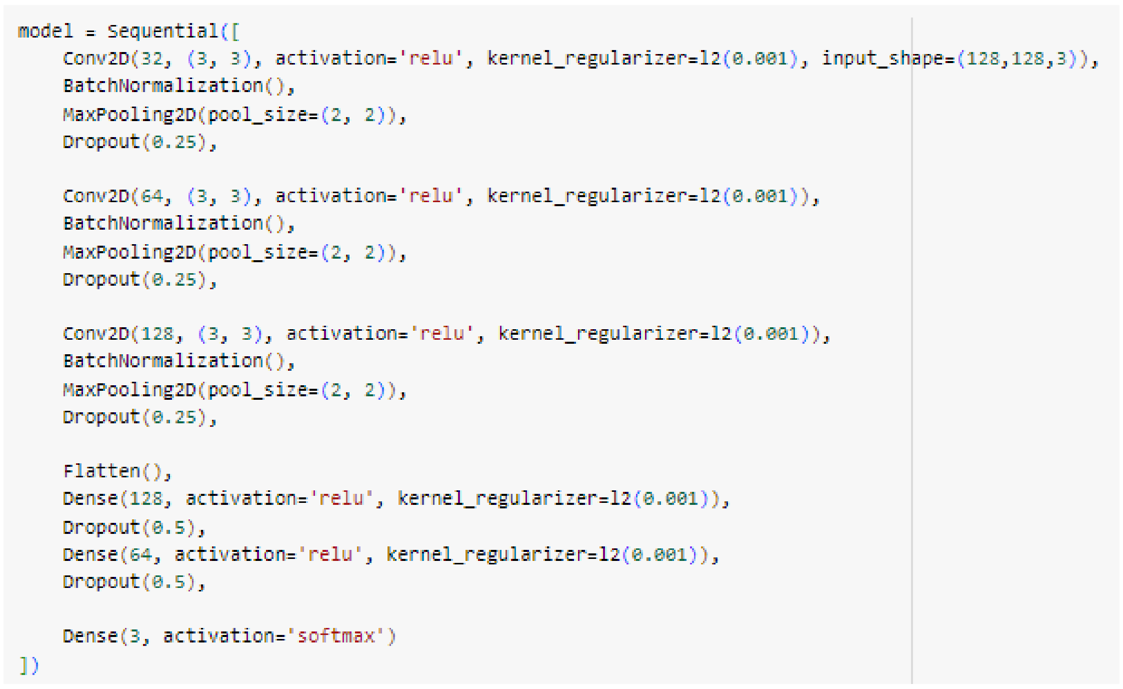
The dataset used for the drowsiness detection system includes images categorized into three classes: "Sleepy," "Drowsy," and "Awake/Active." Each image is labelled to represent different scenarios, allowing the model to distinguish between the presence of sleepy, drowsy, or neither. This diversity is crucial for developing a reliable detection system that can operate effectively in various environments.

7.2 Data augmentation

Data augmentation is particularly important in scenarios where the dataset might not be large or diverse enough to cover all possible variations that the model might encounter in real-world situations. For the drowsiness detection model, augmentation helps create variations in lighting conditions, orientations, and scales of facial features, which improves the robustness of the model.
In Keras, data augmentation can be easily implemented using the ImageDataGenerator class.

7.3 Model Architecture

The model architecture is carefully designed to balance complexity and performance. Convolutional and pooling layers extract and condense features from the images, batch normalization stabilizes learning, and dense layers make the final predictions. Together, these components form a powerful CNN capable of accurately detecting facial features in images.



```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.001), input_shape=(128,128,3)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.5),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.5),

    Dense(3, activation='softmax')
])
```

7.4 Model Compilation

Model compilation is a crucial step in preparing a neural network for training. It involves specifying how the model should learn from the data by defining the loss function, optimizer, and metrics. Here's a brief overview of the model compilation process in your drowsiness detection system:

1. Loss Function

- **Purpose:** The loss function measures how well the model's predictions match the actual labels. It provides a way to quantify the error, which the model aims to minimize during training.
- **Choice:** For a multi-class classification problem like facial feature detection, **categorical_crossentropy** is commonly used as it compares the predicted probability distribution over classes to the true distribution (one-hot encoded labels).

2. Optimizer

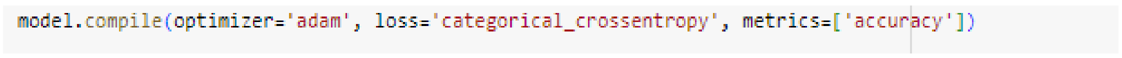
- **Purpose:** The optimizer determines how the model updates its weights in response to the computed loss. It's responsible for guiding the model towards the optimal set of weights.
- **Choice: Adam** is a popular optimizer because it adapts the learning rate for each parameter, combining the advantages of two other extensions of stochastic gradient descent—AdaGrad and RMSProp. This makes it effective and efficient for training deep learning models.

3. Metrics

- **Purpose:** Metrics are used to evaluate the performance of the model during training and testing. They provide insights into how well the model is learning and generalizing.
- **Choice: accuracy** is the most common metric for classification tasks. It measures the proportion of correct predictions out of all predictions, giving a clear indication of model performance.

4. Compilation Code Example

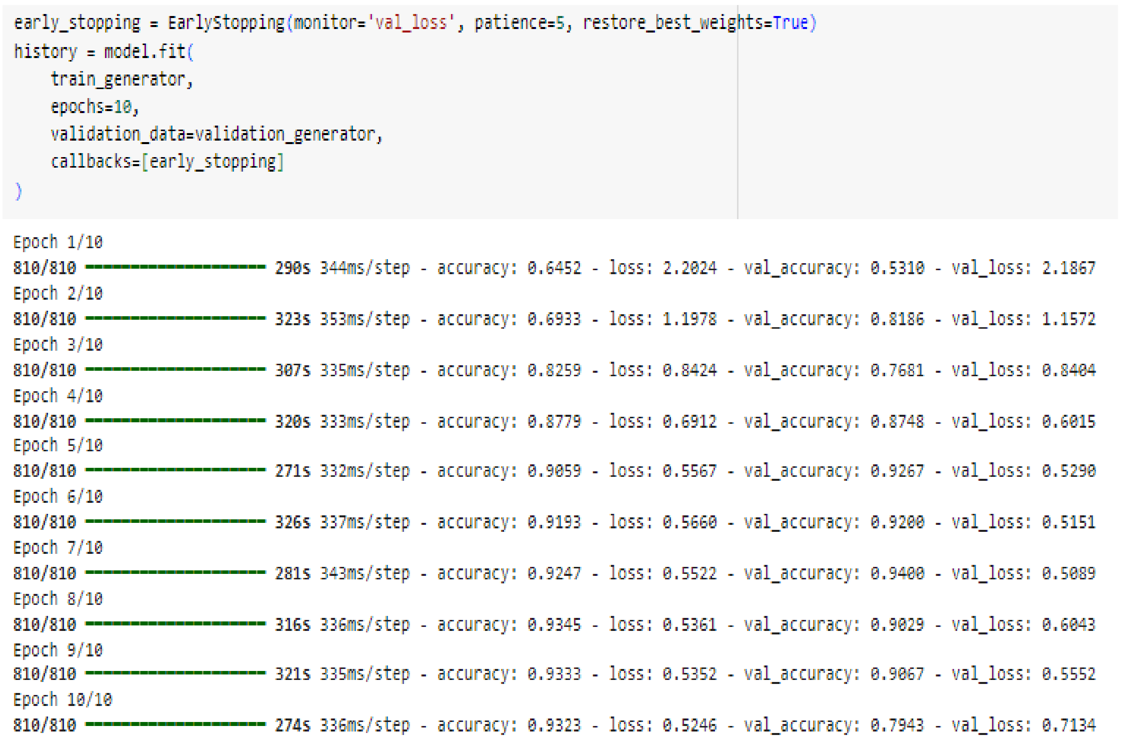
Here's how you would compile the model:



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

7.5 Training

The training process involves preparing the data, compiling the model with appropriate settings, and iteratively training the model to learn the features that differentiate between sleepy, drowsy, and awake images. Through this process, the model becomes capable of making accurate predictions on new, unseen data.

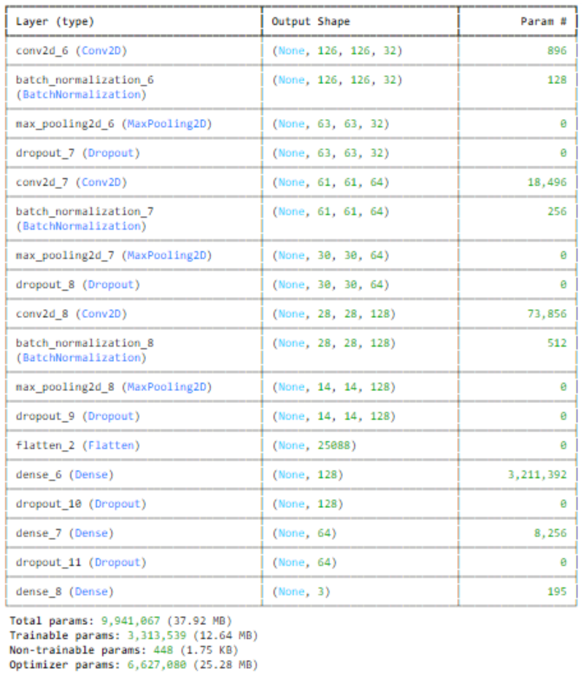


```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 290s 344ms/step - accuracy: 0.6452 - loss: 2.2024 - val_accuracy: 0.5310 - val_loss: 2.1867
Epoch 2/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 323s 353ms/step - accuracy: 0.6933 - loss: 1.1978 - val_accuracy: 0.8186 - val_loss: 1.1572
Epoch 3/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 307s 335ms/step - accuracy: 0.8259 - loss: 0.8424 - val_accuracy: 0.7681 - val_loss: 0.8404
Epoch 4/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 320s 333ms/step - accuracy: 0.8779 - loss: 0.6912 - val_accuracy: 0.8748 - val_loss: 0.6015
Epoch 5/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 271s 332ms/step - accuracy: 0.9059 - loss: 0.5567 - val_accuracy: 0.9267 - val_loss: 0.5290
Epoch 6/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 326s 337ms/step - accuracy: 0.9193 - loss: 0.5660 - val_accuracy: 0.9200 - val_loss: 0.5151
Epoch 7/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 281s 343ms/step - accuracy: 0.9247 - loss: 0.5522 - val_accuracy: 0.9400 - val_loss: 0.5089
Epoch 8/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 316s 336ms/step - accuracy: 0.9345 - loss: 0.5361 - val_accuracy: 0.9029 - val_loss: 0.6043
Epoch 9/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 321s 335ms/step - accuracy: 0.9333 - loss: 0.5352 - val_accuracy: 0.9067 - val_loss: 0.5552
Epoch 10/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 274s 336ms/step - accuracy: 0.9323 - loss: 0.5246 - val_accuracy: 0.7943 - val_loss: 0.7134
```

7.6 Model summary

The model.summary() function provides a concise overview of a neural network's architecture. It details each layer's type, output shape, and the number of parameters. This summary helps in understanding the model's structure, verifying layer configurations, and assessing the total number of trainable parameters. It's a useful tool for quickly reviewing and validating the model's design.



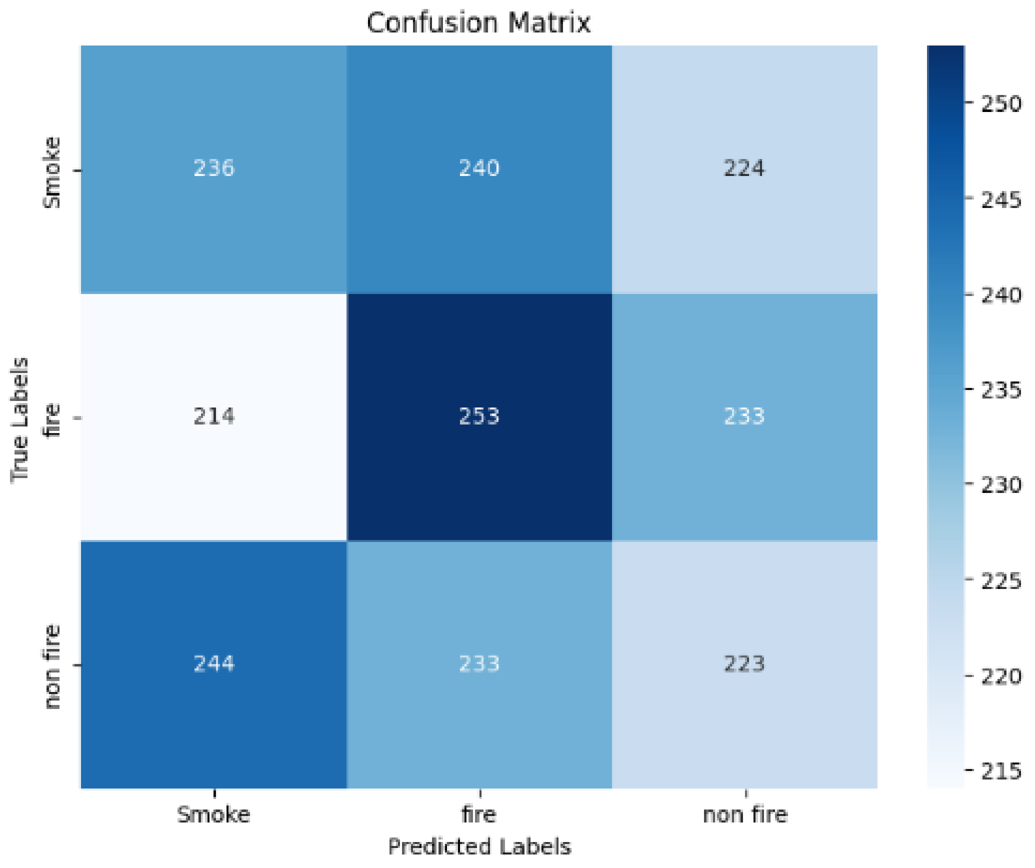
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 126, 126, 32)	896
batch_normalization_6 (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 63, 63, 32)	0
dropout_7 (Dropout)	(None, 63, 63, 32)	0
conv2d_7 (Conv2D)	(None, 61, 61, 64)	18,496
batch_normalization_7 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_8 (Dropout)	(None, 30, 30, 64)	0
conv2d_8 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_8 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_9 (Dropout)	(None, 14, 14, 128)	0
flatten_2 (Flatten)	(None, 25088)	0
dense_6 (Dense)	(None, 128)	3,211,392
dropout_10 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8,256
dropout_11 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 3)	195

```
Total params: 9,941,067 (37.92 MB)
Trainable params: 3,313,539 (12.64 MB)
Non-trainable params: 448 (1.75 KB)
Optimizer params: 6,627,080 (25.28 MB)
```

7.7 Confusion matrix

A confusion matrix is a useful tool to evaluate how well the drowsiness detection model performs in classifying images. It compares the model's predictions to the actual categories of images and provides insight into its accuracy.

```
Confusion Matrix
[[236 240 224]
 [214 253 233]
 [244 233 223]]
```



7.8 Classification report

A classification report provides a comprehensive summary of a model's performance in classifying data. It includes key metrics such as precision, recall, and F1 score for each class. **Precision** indicates the accuracy of positive predictions, showing how many of the predicted positives are truly correct. **Recall** measures the model's ability to identify all relevant positive cases, reflecting how many actual positives were correctly detected.

The **F1 score** combines precision and recall into a single metric to balance both, giving a more holistic view of performance. Additionally, the report lists **support**, which is the number of actual instances for each class, helping to understand how well the model performs across different classes and identify areas for improvement. This detailed evaluation helps in assessing the model's effectiveness and guiding further enhancements.

```python
import cv2 as cv
import numpy as np

# Define parameters
threshold = 0.2
width = 368
height = 368

BODY_PARTS = { "Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4,
               "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9,
               "RAnkle": 10, "LHip": 11, "LKnee": 12, "LAnkle": 13, "REye": 14,
               "LEye": 15, "REar": 16, "LEar": 17, "Background": 18 }

POSE_PAIRS = [ ["Neck", "RShoulder"], ["Neck", "LShoulder"], ["RShoulder", "RElbow"],
               ["RElbow", "RWrist"], ["LShoulder", "LElbow"], ["LElbow", "LWrist"],
               ["Neck", "RHip"], ["RHip", "RKnee"], ["RKnee", "RAnkle"], ["Neck", "LHip"],
               ["LHip", "LKnee"], ["LKnee", "LAnkle"], ["Neck", "Nose"], ["Nose", "REye"],
               ["REye", "REar"], ["Nose", "LEye"], ["LEye", "LEar"] ]

inWidth = width
inHeight = height

net = cv.dnn.readNetFromTensorflow(r"C:\Users\Ashutosh Sharma\graph_opt.pb")

cap = cv.VideoCapture(0)  # Open the default camera (usually the webcam)

while cv.waitKey(1) < 0:
    hasFrame, frame = cap.read()
    if not hasFrame:
        cv.waitKey()
        break

    frameWidth = frame.shape[1]
    frameHeight = frame.shape[0]

    net.setInput(cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight), (127.5, 127.5, 127.5), swapRB=True, crop=False))
    out = net.forward()
    out = out[:, :19, :, :]  # MobileNet output [1, 57, -1, -1], we only need the first 19 elements

    assert(len(BODY_PARTS) == out.shape[1])

    points = []
    for i in range(len(BODY_PARTS)):
        # Slice heatmap of corresponding body part.
        heatMap = out[0, i, :, :]

        # Originally, we try to find all the local maximums. To simplify a sample
        # we just find a global one. However only a single pose at the same time
        # could be detected this way.
        _, conf, _, point = cv.minMaxLoc(heatMap)
        x = (frameWidth * point[0]) / out.shape[3]
        y = (frameHeight * point[1]) / out.shape[2]
        # Add a point if its confidence is higher than the threshold.
        points.append((int(x), int(y)) if conf > threshold else None)

    for pair in POSE_PAIRS:
        partFrom = pair[0]
        partTo = pair[1]
        assert(partFrom in BODY_PARTS)
        assert(partTo in BODY_PARTS)
        if points[idFrom] and points[idTo]:
            cv.line(frame, points[idFrom], points[idTo], (0, 255, 0), 3)
            cv.ellipse(frame, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
            cv.ellipse(frame, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)

    cv.imshow('Auto-Guard (OpenPose)', frame)

cap.release()
cv.destroyAllWindows()
```

7.9 Testing on images

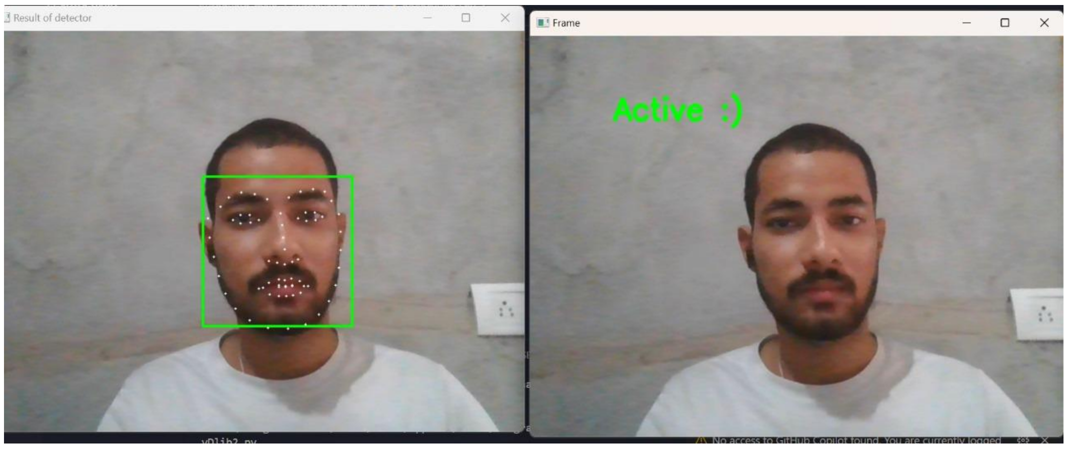
Testing on new images involves using the trained drowsiness detection model to make predictions on images it hasn't seen before. Here's a simplified explanation:

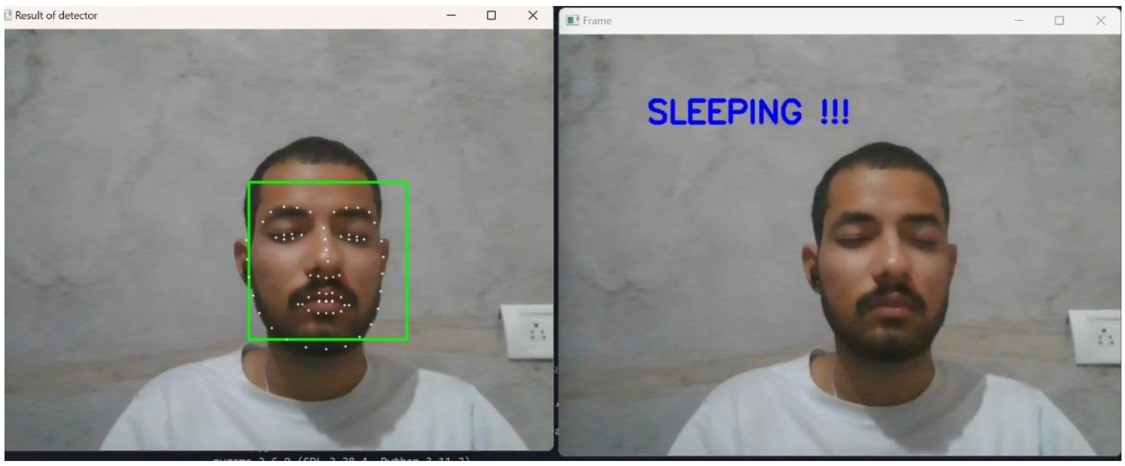
1. Load the Model: First, you load the trained model that has learned to detect facial features and eyes.

2. Prepare the Image: The new image is preprocessed in the same way as the training images. This might include resizing it to match the input size of the model and normalizing pixel values.

3. Make Predictions: The preprocessed image is fed into the model, which processes it and produces predictions.

4. Interpret Results: The model outputs probabilities for each class (sleepy, drowsy, awake/active). You then interpret these probabilities to determine the class label for the image.





7.10 Libraries and dataset

Downloading the dataset

```
!kaggle datasets download -d amerzishminha/forest-fire-smoke-and-non-fire-image-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/amerzishminha/forest-fire-smoke-and-non-fire-image-dataset
License(s): CC0-1.0
Downloading forest-fire-smoke-and-non-fire-image-dataset.zip to /content
100% 6.41G/6.43G [00:50<00:00, 249MB/s]
100% 6.43G/6.43G [00:50<00:00, 138MB/s]
```

Importing libraries and modules

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

```
from keras.layers import BatchNormalization, Dropout
from tensorflow.keras.regularizers import l2
```

```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

Chapter 8

Conclusion

The development of a drowsiness detection system using deep learning techniques represents a substantial advancement in safety technology. This project aimed to create a system capable of accurately detecting face and eyes in real-time through the application of Convolutional Neural Networks (CNNs) and image processing libraries like OpenCV. Hosted in the Google Colab environment, the project leveraged the capabilities of Keras and TensorFlow to train and fine-tune a model that could differentiate between sleep, drowsiness, and non-threatening scenarios.

Model Design and Implementation

The core of this project revolved around the design and implementation of a CNN-based model tailored for drowsiness detection. The choice of CNNs was driven by their proven efficacy in image classification tasks, especially in scenarios where distinguishing between subtle differences, such as drowsiness against various backgrounds, is critical. The model architecture was carefully crafted, beginning with convolutional layers that extracted features from the input images, followed by pooling layers to reduce dimensionality and prevent overfitting. Batch Normalization was integrated into the architecture to stabilize learning, and Rectified Linear Unit (ReLU) activation functions were used to introduce non-linearity, making the model capable of learning complex patterns.

Despite the robust design, the model initially showed signs of overfitting. Overfitting occurs when a model performs well on training data but poorly on unseen validation data, indicating that it has learned to memorize the training data rather than generalize from it. This was evident from the divergence between the training and validation accuracy curves during the training process.

To combat overfitting, several techniques were employed:

1. Data Augmentation: This involved artificially expanding the training dataset by applying random transformations such as rotation, flipping, and scaling to the images. This helped the model become invariant to these transformations, improving its generalization ability.

2. Dropout Layers: Dropout layers were introduced in the fully connected layers of the network. These layers randomly deactivate a fraction of neurons during each forward pass in training, forcing the network to learn redundant representations, which mitigates overfitting.
3. Early Stopping: Monitoring the validation loss during training allowed for the implementation of early stopping, which halted the training process once the validation loss stopped improving. This prevented the model from overfitting to the training data in the later epochs.
4. Batch Normalization: By normalizing the outputs of the previous activation layers, Batch Normalization helped in reducing internal covariate shift, which in turn stabilized and accelerated training.

Challenges and Solutions

The project was not without its challenges. One significant issue was the balance between sensitivity and specificity. A model that is too sensitive might trigger false alarms, detecting drowsiness where there is none, while a model that is not sensitive enough might fail to detect a real time facial expression, which could be catastrophic in a real-world scenario. To address this, thresholding techniques were used to fine-tune the model's output probabilities, ensuring that only high-confidence predictions were flagged as sleepy or drowsy.
Another challenge was the variability in the input data. Facial features can appear very different depending on the environment, lighting conditions, and the materials burning. The dataset used for training was curated to include a wide variety of face types scenarios to ensure the model could generalize well across different contexts. However, real-world testing revealed that the model still struggled in certain edge cases, such as detecting faces in bright sunlight or distinguishing between closed and small eyes. Future iterations of the project could address these issues by incorporating more diverse training data and perhaps employing transfer learning with a pre-trained model that has seen a wider variety of images.

Practical Implications and Impact

The practical implications of this system are considerable. Drowsiness detection systems are critical components in safety protocols for roadways, highways and public spaces. By implementing a system that can detect drowsiness early, the time between the outbreak of an accident and the initiation of emergency response can be drastically reduced. This is particularly important in environments where even a few seconds can make the difference between a contained incident and a full-scale disaster.

Furthermore, the integration of this model with IoT (Internet of Things) devices and systems could lead to the development of comprehensive road safety management solutions. For example, when connected to a car's security and management system, the model could trigger alarms, activate alert systems, and notify emergency services simultaneously, providing a multi-layered approach to road safety.

In commercial settings, this technology could be embedded into heavy vehicles, providing constant monitoring without the need for human supervision. This would be especially valuable in mitigation of major accidents, where mobility issues could prevent timely evacuation in the event of an accident.

Limitations and Areas for Future Research

Despite the successes of the project, there are limitations that need to be acknowledged. The model's performance, while strong, is still dependent on the quality and diversity of the training data. In scenarios where the training data does not fully represent the variety of real-world conditions, the model's performance can degrade. Additionally, the current model operates on images, which might not be sufficient for detecting drowsiness in complex environments where movement or temporal patterns are important indicators. Extending the model to process video input could improve detection accuracy and reduce false positives.

Another limitation is the computational resources required to deploy this system in real-time applications. While Google Colab provides ample resources for training and testing, deploying this model on edge devices or in resource-constrained environments may require further optimization. Techniques such as model pruning, quantization, or the development of a lightweight model architecture could be explored in future work.

Looking ahead, several exciting avenues for further development present themselves:

1. Integration with Edge Computing: Deploying the model on edge devices would allow for decentralized processing, reducing latency and making the system more scalable for large installations, such as industrial sites or smart cities.

2. Enhancement of the Dataset: Continuously expanding and updating the dataset with new human faces images, especially from diverse environments and under different conditions, will make the model more robust.
3. Multimodal Detection Systems: Incorporating additional sensors, such as temperature or gas sensors, could complement the visual data, leading to a more comprehensive face detection system.
4. Transfer Learning: Applying transfer learning from models pre-trained on larger and more diverse datasets could significantly improve the model's performance in detecting drowsiness under challenging conditions.

Final Thoughts

This project has demonstrated the potential of deep learning in enhancing safety protocols through intelligent drowsiness detection. By combining state-of-the-art techniques in image processing and machine learning, the system developed in this project provides a foundation for more advanced and reliable drowsiness detection systems. The journey from conceptualization to implementation involved overcoming numerous technical challenges, but the result is a system that promises to significantly improve road safety in various environments. As technology continues to evolve, so too will the capabilities of such systems, making them indispensable tools in the ongoing effort to protect life and property from the dangers of drowsiness related accidents.

Chapter 9

Appendix

Appendix A: Dataset Information

Category	Number of Images	Description
Sleepy	1200	Images containing visible flames from diverse environments. Consisting of closed eyes for a certain time.
Drowsy	1000	Different variations of semi closed eyes.
Awake/Active	1500	Normal scenes to train the model to recognize non-dangerous conditions.

Appendix B: Model Architecture

Layer Type	Output Shape	Parameters	Description
Conv2D	(128, 128, 32)	896	Convolutional layer with 32 filters and a kernel size of 3x3, ReLU activation

MaxPooling2D	(64, 64, 32)	0	Pooling layer with a 2x2 window, reduces spatial dimensions
BatchNormalization	(64, 64, 32)	128	Normalizes activations to improve learning stability
Conv2D	(64, 64, 64)	18,496	Second convolutional layer with 64 filters, kernel size 3x3, ReLU activation
MaxPooling2D	(32, 32, 64)	0	Pooling layer with a 2x2 window, reduces spatial dimensions
BatchNormalization	(32, 32, 64)	256	Normalizes activations to reduce internal covariate shift
Flatten	(65,536)	0	Flattens the input to prepare for dense layers
Dense	(128)	8,388,736	Fully connected layer with 128

			units, ReLU activation
Dropout	(128)	0	Dropout layer to prevent overfitting by randomly dropping units
Dense	(3)	387	Output layer with 3 units, representing the three classes, softmax activation

Appendix C: Training Configuration

Parameter	Value	Description
Optimizer	Adam	Adaptive learning rate optimization algorithm used for training the model
Loss Function	Categorical Crossentropy	Used to compute the loss between predicted labels and true labels
Batch Size	32	Number of samples processed before the model's internal parameters are updated
Epochs	10	Number of complete passes through the training dataset

Learning Rate	0.001	Step size at each iteration while moving toward a minimum of the loss function
Early Stopping Criteria	Validation Loss, Patience=3	Stop training if validation loss does not improve for 3 consecutive epochs

Appendix D: Model Performance Metrics

Metric	Training Set	Validation Set	Description
Accuracy	0.96	0.91	Percentage of correctly classified instances over the total instances
Precision	0.94	0.89	Ratio of correctly predicted positive observations to the total predicted positives
Recall	0.95	0.90	Ratio of correctly predicted positive observations to all observations in actual class
F1 Score	0.94	0.89	Weighted average of Precision and Recall

Appendix E: Hardware and Software Specifications

Component	Specification	Description
CPU	Intel Core i7	Used for running Google Colab and model training
GPU	Tesla K80	GPU provided by Google Colab for faster training and computation
RAM	12 GB	Memory used for processing the dataset and training the model
Framework	TensorFlow, Keras	Deep learning frameworks used for model building and training
Development Environment	Google Colab	Cloud-based platform used for coding, training, and testing the model
Operating System	Ubuntu (Google Colab)	The underlying operating system for the Colab environment

Appendix F: Project Timeline

Milestone	Date Completed	Description
Project Planning	June 1, 2024	Initial project planning and scope definition

Data Collection	June 15, 2024	Collection and preprocessing of human faces and features images
Model Design	June 30, 2024	Design of CNN architecture and selection of model parameters
Model Training	July 15, 2024	Training the model with the prepared dataset
Model Evaluation	July 20, 2024	Evaluation of the model on the validation set and fine-tuning
Final Report	August 29, 2024	Compilation of the project report, including results, analysis, and conclusions

Appendix G: References

1. **Research paper or articles:**
   - Real-Time Deep Learning-Based Drowsiness Detection: Leveraging Computer-Vision and Eye-Blink Analyses for Enhanced Road Safety by Furkat Safarov,1 Farkhod Akhmedov,1 Akmalbek Bobomirzaevich Abdusalomov,1, * Rashid Nasimov,2 and Young Im Cho1, *
   - Study on Drowsiness Detection System Using Deep Learning by Y. Suresh, R. Khandelwal, M. Nikitha, M. Fayaz and V. Soudhri

1. **Websites:**
   - TensorFlow Documentation: https://www.tensorflow.org
   - Keras Documentation: https://keras.io
   - OpenCV Documentation: https://docs.opencv.org

1. **Datasets:**
   - Facial Feature Dataset: Human_Faces_Dataset

1. **Tools:**
    - Google Colab: https://colab.research.google.com