**Detailed Report**

Title: Driver Drowsiness Detection using Deep Learning

**1. Problem Specification and Motivation:**

Driving while feeling drowsy is a major problem that contributes to a significant percentage of traffic accidents worldwide. According to various studies, fatigue-related incidents account for about 20% of all road accidents, rising to as much as 50% on certain roads. As technology advances, innovative solutions are being sought to mitigate this issue. Autonomous or self-driving cars employ an array of sensors, cameras, and artificial intelligence to traverse without human operators. One such feature is driver drowsiness detection, which prevents accidents caused by driver fatigue.

Our project aims to leverage the power of AI, specifically deep learning, to predict driver drowsiness based on facial expressions and eye movements. The dataset includes images in various categories, such as 'Eyes Closed', 'Eyes Open', 'Yawn', and 'No Yawn'. Our team will use this data to train a deep learning model capable of accurately predicting driver drowsiness and issuing warnings accordingly. This project represents an applied problem that has immense societal and safety implications.

**2. Methodology Used:**

We propose to use a combination of computer vision and deep learning methods for this project. A convolutional neural network (CNN) will be the backbone of our deep learning model. CNNs are particularly suited for image recognition tasks as they can capture spatial hierarchies and patterns in images.

First, we'll preprocess the images from the dataset, normalizing and resizing them as necessary. Then, we'll create a CNN model, which we'll train on our dataset. We'll make use of transfer learning, starting with a pretrained model such as VGG16 or ResNet50, which we'll then fine-tune to our specific task. We'll employ methods like dropout and batch normalization to prevent overfitting.

Furthermore, we will consider employing machine learning techniques such as Support Vector Machines (SVM) or Random Forests as a comparison benchmark to gauge the performance of our deep learning model.

**3. Theory and Dataset:**

The dataset sourced from Kaggle, titled "Drowsiness Dataset", is geared towards the detection of drowsiness in individuals.

**Structure:** The dataset contains images segmented into various classes that denote the state of drowsiness, specifically: 'Yawn', 'No\_yawn', 'Open', and 'Closed'. These labels are indicative of mouth and eye states, vital indicators of drowsiness.

**Yawn and No\_Yawn:** These classes are straightforward. 'Yawn' contains images where individuals are yawning, and 'No\_Yawn' has images where they aren't.

**Open and Closed:** These classes denote the state of eyes. 'Open' suggests the person's eyes are open, suggesting alertness, while 'Closed' might hint towards fatigue or drowsiness.

**Variability:** The dataset embraces a wide range of scenarios. There are images under various lighting conditions, multiple angles, diverse facial structures, and more. Such variability ensures that the trained model doesn't overfit to specific conditions and generalizes well to unseen data.

**Utility:** The dataset, by focusing on both mouth and eye states, provides a holistic approach to drowsiness detection. While yawning is a direct indicator of fatigue, the state of the eyes can be a subtle, yet crucial, hint towards the onset of drowsiness.

**Source and Licensing:** This dataset is available on Kaggle, a popular platform for machine learning and data science projects. Before using it for commercial applications, always ensure you've the right permissions and are aware of any licensing restrictions.

https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset

**4. Implementation:**

**1. Environment Setup and Pre-requisites:**

The implementation starts with importing the necessary libraries that act as the backbone for the project.

**TensorFlow and Keras:** The project's foundation is the TensorFlow framework with Keras as its high-level API. They provide the tools and modules necessary for building and training deep learning models.

**OpenCV (cv2):** Used for image processing tasks, from reading images to transforming them.

Matplotlib, Seaborn, and Plotly: For visualization purposes, whether it's plotting graphs or displaying images.

**Sklearn:** Provides useful tools for splitting the dataset and evaluating the model.

**2. GPU Check and Image Size Definition:**

A quick check is made to determine if a GPU is available for computation, enhancing the efficiency of training. The constant IMAGE\_SIZE is also defined to set a standard size for all processed images.

**3. Data Preparation - 'datamaker' function:**

This function processes the data:

Images are read and converted from BGR to RGB since OpenCV reads images in BGR format by default.

They're resized to the predefined IMAGE\_SIZE and normalized (scaled between 0 and 1).

The data is labeled based on the directory it is fetched from, making this function both efficient and scalable.

**4. Data Splitting:**

The dataset is split into three parts:

**Training Set:** Used for training the model.

**Validation Set:** Used to tune the model parameters without touching the test data.

**Test Set:** Used for evaluating the final model.

This separation ensures that the model doesn't overfit and can generalize well on unseen data.

**5. Data Augmentation:**

Using ImageDataGenerator, data augmentation is applied only on the training set to artificially enhance its size and variability. Techniques like rotation, zooming, and horizontal flipping are applied to make the model robust against slight variations in the input.

**6. Transfer Learning - InceptionV3 Model:**

The pre-trained InceptionV3 model is integrated:

Its top layers are excluded to customize it for our 4-class problem.

The model's earlier layers are frozen, ensuring the weights (learned features from ImageNet) remain unchanged during training. This maximizes the benefits of transfer learning.

**7. Model Customization and Compilation:**

On top of the base InceptionV3 model, several layers are added:

A Dense layer with ReLU activation to learn intricate patterns.

Flatten to reshape the model's output, making it suitable for the final classification layer.

Dropout for regularization, reducing chances of overfitting.

A final Dense layer with softmax activation for classification into four classes.

The model is then compiled with the 'adam' optimizer and 'sparse categorical crossentropy' as the loss function, suitable for multi-class labeled datasets.

**8. Training:**

The model is trained on the augmented training data, with validation data providing feedback for 10 epochs. The training process's progress is tracked, including accuracy and loss metrics for both training and validation sets.

**9. Evaluation and Visualization:**

Post-training, various metrics (accuracy, loss, precision, recall, F1-score) are computed. Confusion matrix and classification reports are generated to offer insights into the model's performance class-wise. Additionally, some test samples are visualized alongside their true and predicted labels.

**10. Model Persistence:**

The fully trained model is saved as "group\_transferlearning.h5", allowing for its reuse without retraining, a significant time saver.

**5. Explanation of the Source code**

**Details on Library Importation**

At the outset, we imported a suite of essential libraries. These libraries facilitate tasks ranging from numerical operations (numpy) to visualization (matplotlib and seaborn), OS-level operations (os), deep learning processes (tensorflow), and image processing (cv2).

**GPU Configuration and Setup**

We instituted a GPU check, defaulting to the CPU if none was available. This approach ensures that we harness the best available hardware for neural network training.

A black screen with white text

Description automatically generated

**Data Preparation**

We established IMAGE\_SIZE as 256, denoting our commitment to resizing all images to 256x256 pixels.

In our datamaker() function, images are procured, formatted to the aforementioned size, and subsequently labeled according to "yawn," "no\_yawn," "Open," or "Closed" categories.

Having collated this dataset, we randomized the order and divided it into distinct features (X) and labels (Y).

|  |  |
| --- | --- |
|  |  |

**Data Splitting**

Further division was executed, breaking the data into training, validation, and testing sets. This ensures our model's training, its performance validation, and eventual testing on previously unseen data.

**Data Augmentation**

We incorporated ImageDataGenerator for data augmentation on the training dataset. Implementing rotations, zooming, and horizontal flipping, this enhancement diversifies the training dataset, minimizing potential overfitting.

**Transfer Learning with InceptionV3**

Tapping into the power of the pre-trained InceptionV3 model, trained on the imagenet dataset, we transferred its learned features for our project.

The initial 230 layers of this model were "frozen" (or set untrainable). The subsequent layers, inclusive of the ones we appended, were calibrated in accordance with our dataset.

We then enriched the InceptionV3 model with additional dense layers, dropout layers for regularization, and a final classification layer with softmax activation.

Training the Model

After defining the model, we compiled it with the Adam optimizer, targeting the sparse categorical cross-entropy loss and monitoring the accuracy metric.

The model was trained using the augmented training set and validated via the validation dataset.

**OWN MODEL**

A graph with blue and orange lines

Description automatically generated

A graph with blue and orange lines

Description automatically generated

A number on a white background

Description automatically generated

Graph mapping training and validation accuracy alongside loss metrics over the epochs.

**USING INCEPTION MODEL**

A graph of a graph

Description automatically generated with medium confidence

A graph with blue line and orange line

Description automatically generated

A number with numbers on it

Description automatically generated with medium confidence

Graph mapping training and validation accuracy alongside loss metrics over the epochs.

**Results and Evaluation**

Subsequent to training, we gauged the model's prowess on the testing set.

Additionally, a confusion matrix was charted to evaluate the model's predictions vis-à-vis the actual labels.

A blue and red squares with numbers

Description automatically generated

A heatmap representation of the confusion matrix of **OWN MODEL**

A blue and red squares with white text

Description automatically generated

A heatmap representation of the confusion matrix of **INCEPTION MODEL**

A screenshot of a report

Description automatically generated

A classification report of **INCEPTION MODEL**

**6. Results and Discussion:**

When the trained model was evaluated on the test set, the results were consistent with the validation performance, further solidifying the robustness of our approach.

Visualization tools, such as accuracy and loss plots, provided valuable insights into the model's learning trajectory. These visual aids illustrated that the model's training and validation accuracy converged, indicating minimal overfitting. The confusion matrix, which was rendered to understand the model's performance across different classes, further demonstrated the model's proficiency in distinguishing between the classes 'Yawn', 'No\_yawn', 'Open', and 'Closed'.

The accompanying classification report, which provided metrics like precision, recall, F1-score, and AUC, further supported the high-level accuracy figures, showcasing the model's adeptness in handling individual classes.

A number on a white background

Description automatically generated

A computer screen shot of a computer code

Description automatically generated**Achieved Accuracy of 0.997**A collage of a person in a car

Description automatically generated

**OWN MODEL RESULTS**

A collage of images of a person in a car

Description automatically generated

**USING INCEPTION MODEL RESULTS**

A few demonstrative images of real and predicted labels using.

**Model Preservation**

Concluding our efforts, we secured the trained model in the group\_transferlearning.h5 file, facilitating its future deployment without necessitating retraining.

Upon completing the training process using the transfer learning approach with the InceptionV3 architecture, impressive results were achieved. By the 10th epoch, the model reported an accuracy of 99% on the validation set, which stands as a testament to the power and efficiency of transfer learning.

**7. Recommendations for Future work**

While the current implementation yields promising results, there's always room for improvement and exploration:

**Data Augmentation Techniques**: Expanding upon the current data augmentation methods may enhance the model's generalization capabilities. Techniques such as vertical flips or brightness variations could be explored.

**Experimentation with Different Architectures:** While InceptionV3 was effective, other architectures like ResNet, EfficientNet, or VGG might offer varied performance metrics and should be considered in future iterations.

**Hyperparameter Tuning:** A deeper dive into hyperparameter optimization, possibly using tools like Keras Tuner or GridSearchCV, might lead to further improvements in model performance.

**Ensemble Methods:** Combining predictions from multiple models, or using techniques like stacking, could improve the robustness and accuracy of predictions.

**Real-time Application:** Future work could also involve integrating the model into a real-time drowsiness detection system, enhancing its practical application for safety-critical operations like driving.

**Feedback Loop:** As the model is deployed and used, collecting feedback and misclassified samples can help in further refining and retraining the model for even better accuracy in subsequent iterations.