Drowsiness and Emotion Monitor

Joanna Findura, Allan Watatani and Furkan Mohamed
Department of Electrical and Computer Engineering I
lllinois Institute of Technology, Chicago IL, U.S.A

***Abstract*—This paper presents a Drowsiness and Distraction Detection System using computer vision and machine learning. The system aims to detect and alert drivers displaying signs of drowsiness or distraction, crucial factors contributing to road accidents. It leverages the Jetson Nano platform, integrating PyTorch and OpenCV libraries, and runs on a customized Ubuntu OS. A deep learning model trained on a dataset of open and closed eyes is deployed on the Jetson Nano, capturing real-time video feed from a vehicle-installed camera. Computer vision techniques, like Haar cascades, extract eye regions for classification. Additional features include Grad-CAM visualization and an audio alarm. The implemented system shows promising results, accurately identifying driver drowsiness and distraction in real-time, enhancing automotive safety.**

Keywords—Automated Emotion and Drowsiness System, Artificial Intelligence, Real-Time Detection

I. Introduction

In recent years, the development of advanced driver assistance systems (ADAS) has received considerable attention as a means to improve road safety and reduce accidents caused by driver errors. A critical aspect of ADAS is the detection of driver drowsiness and distraction, which significantly contribute to road accidents. This project aimed to design and implement a real-time system for detecting drowsiness and distraction using computer vision techniques and deep learning algorithms. The primary objective was to create an efficient and robust system capable of monitoring the driver's eye behavior and providing real-time alerts when signs of drowsiness or distraction were detected. To achieve this, we utilized the Jetson Nano, an embedded platform known for its computational capabilities and energy efficiency. The system integrated various software components, including PyTorch, OpenCV, and Python, all tailored to the Ubuntu OS designed for the Jetson Nano. Our approach involved employing Haar cascade classifiers from OpenCV to detect faces and eyes, followed by eye state classification using a pre-trained convolutional neural network (CNN) model. Additionally, we incorporated the Grad-CAM technique, which provided visualizations highlighting the regions of the image that contributed most to the classification results, thus offering interpretability to the system's outputs.

II. Drowsiness

A. Dataset

In our project, we utilized a pre-existing model named "BlinkModel.t7," which serves as the foundation for all our tests. However, it is also possible for us to train the model ourselves using the provided code.

The database used for training consists of a collection of 4,846 images of left and right eyes, each captured in both open and closed states. Approximately half of the images depict open eyes, while the other half represent closed eyes. This diverse dataset enables the network to effectively recognize and classify the different eye states. For further details, please refer to the appendix, where the database is located.

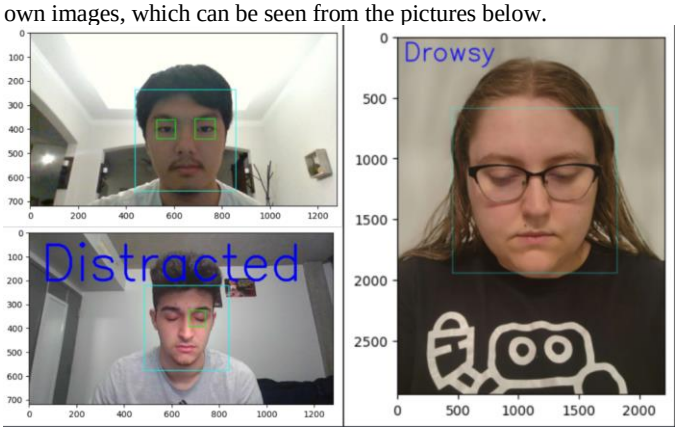
B. Training

To train our model, we developed a Python script that allows us to train the model according to our specific requirements. Before starting the training, we ensured that the database containing our training images was unzipped. This database consists of 4,846 images capturing both left and right eyes in open and closed states. This diverse dataset is vital for training the model to accurately distinguish between the different eye states. The script begins by defining the training parameters. One of the important parameters is the input image shape, which is set to (24, 24). This means that the input images are resized to a resolution of 24 pixels by 24 pixels. This choice balances computational efficiency with capturing relevant information from the images. To monitor the model's performance during training, we allocate a portion of the training data for validation. A validation ratio of 0.1 is utilized, which reserves 10% of the data for validation purposes. This subset is not used for training but serves to evaluate the model's performance and prevent overfitting. During each training iteration, a batch size of 64 is utilized. This means that the model processes 64 images simultaneously, optimizing the training process for efficiency and memory utilization. The model is trained for a specified number of epochs, which in this case is set to 40. Each epoch represents one complete pass through the entire training dataset, allowing the model to gradually learn and improve its performance. The learning rate, set to 0.001, plays a crucial role in training. It determines the step size at which the model's parameters are updated during optimization. A lower learning rate ensures more precise updates but may require more training time to converge. The chosen loss function for training is the cross-entropy loss. This loss function quantifies the dissimilarity between the predicted probabilities and the true labels of the images. By minimizing this loss, the model is encouraged to make accurate predictions. For optimization, the script employs the Adam optimizer. Adam is a popular choice in deep learning due to its adaptive learning rate approach, which adjusts the learning rate individually for each parameter. This adaptive mechanism enhances the training process and helps the model converge more effectively.

C. Testing

During the testing phase of our model for detecting a person's face behind the wheel, we employ a systematic approach. Using the dataset that we trained our model on, we first utilize a face detection algorithm to identify the presence of a face. If a face is detected, we proceed to locate the eyes within the facial region using an eye detection algorithm. Once the eyes are successfully located, we extract them from the image or frame. These extracted eye images, sourced from the same dataset used for training, are then fed into our trained convolutional PyTorch network, which specializes in detecting the state of the eyes. We analyze the eyes to determine whether they are open or closed, allowing us to assess the driver's level of alertness. To prioritize safety, our system activates an alert if both eyes are detected as closed for at least 2 seconds continuously. This alert indicates that the driver may be falling asleep or losing focus, as keeping the eyes on the road is a vital rule of safe driving. Furthermore, we incorporate a second layer of security by monitoring the duration of continuous blinking. If the person blinks continuously

for more than 2 or 3 seconds (according to our criteria), it signifies potential drowsiness or fatigue. This comprehensive testing process, utilizing our trained dataset, enables us to accurately detect signs of drowsiness or inattentiveness, ensuring the safety of the driver and promoting responsible driving habits. We tested the script with our own images, which can be seen from the pictures below.



III. Emotion

A. *Dataset*

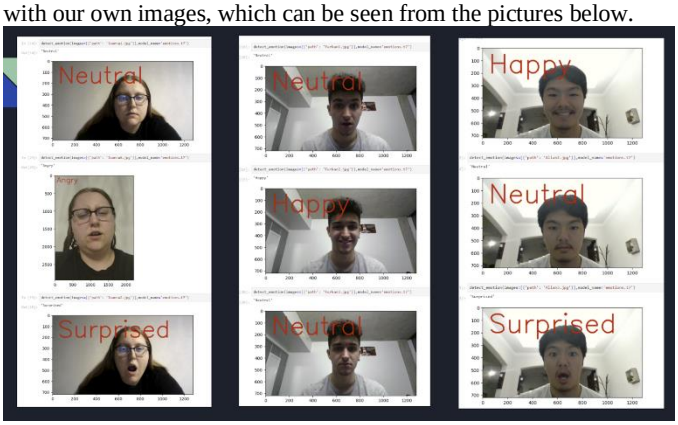
To train our model, we used the FER-2013 dataset on Kaggle. The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

B. *Training*

To begin training, we created a python script that performs a series of essential steps to train and optimize the model, aiming to achieve accurate emotion classification. First, the code prepares the dataset by loading the images and their corresponding emotion labels from the training directory of the FER2013 dataset. It ensures compatibility and simplifies the data by converting the images to grayscale. Additionally, the code maps the emotion categories to numerical labels, enabling efficient training and evaluation. The code further defines the main function, which orchestrates the model training and optimization process. It initializes crucial variables, including the batch size, learning rate, and number of epochs, which play a significant role in controlling the training dynamics. The model's architecture is constructed based on the desired number of output classes, which correspond to different emotion categories to be recognized. During the optimization phase, the code selects an optimizer, typically Stochastic Gradient Descent (SGD) with momentum, and a suitable loss criterion, such as Cross Entropy Loss. It proceeds to iterate over the dataset for the specified number of epochs, reflecting the desired number of complete passes through the training data. Within each epoch, the code dynamically adjusts the learning rate based on a predefined schedule, allowing for adaptive optimization. It processes the dataset in batches, applying forward and backward passes to compute the gradients and update the model's parameters. The training loss and accuracy are calculated as indicators of the model's performance during each epoch. An important aspect of the code is the ability to save the model's state if the current training loss achieves a new minimum validation loss. This feature ensures that the best performing model is preserved for future use, preventing overfitting and promoting generalization.

C. *Testing*

After getting the .t7 file from training the data, we created a Jupiter notebook script to test the model. In the scipt, the architecture of the model is defined, consisting of convolutional layers, batch normalization layers, and pooling operations. Advanced techniques like depth wise separable convolutions and residual blocks are incorporated to enhance the model's performance. For emotion detection, the integration of the OpenCV library allows for face detection using a Haar cascade classifier. The input images are preprocessed by resizing and converting them to grayscale. These processed images are then inputted into the pre-trained emotion detection model. During the inference stage, the trained model is applied to predict the emotion label for each input image. The predicted labels are associated with a predefined list of emotion classes. Additionally, a function is provided to visualize the input image together with the predicted emotion label. We tested the script with our own images, which can be seen from the pictures below.



IV. References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.