Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 242 (2024) 536–543

Procedia Computer Science

www.elsevier.com/locate/procedia

11th International Conference on Information Technology and Quantitative Management (ITQM 2024)

Real-time Tiredness Detection System using Nvidia Jetson Nano and OpenCV

Nicolae Florian[a]* , Prof. PhD. Dumitru Popescu[a,b], Prof. Phd. Andrei Hossu[a]

a. Department of Automatic Control and Systems Engineering, University Politehnica Bucuresti, Bucharest, 060042, Romania

b. Academy of Romanian Scientists, Romania

Abstract

In this paper, we explore how to use a Nvidia Jetson Nano and Python to create a system that detects weariness in a person's face using computer vision and machine learning techniques. The system captures the person's face in real time, preprocesses the picture to cut noise, then detects the face using Haar cascades. Next, using computer vision algorithms, characteristics associated with weariness, such as the eyes, are retrieved. These characteristics are then used to build a machine learning model that can predict weariness in the live stream feed. Lastly, using a graphical user interface, the findings are shown, and the system may be fine-tuned to increase accuracy. This device might be used in applications such as driver monitoring and traffic safety.

© 2024 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 11th International Conference on Information Technology and Quantitative Management

Keywords: Neural Network, Machine Learning, Computer Vision, Driver Monitoring.

1. Introduction

Tiredness or fatigue is a widespread problem that affects many people in various settings, such as driving. Tiredness can impair cognitive and motor skills, reduce alertness and concentration, and increase the risk of accidents and errors. Fatigue can be caused by a few circumstances, including a lack of sleep, sickness, or medicine, and it can impair a person's ability to complete everyday tasks. One of the most common causes of accidents is driver lethargy and weariness. Each year, the number of people killed in such incidents rises across the world. The purpose of this study is to reduce the frequency of accidents caused by driver sleepiness and exhaustion. As a result, transportation safety will improve. Driver drowsiness detection is a car technology that can help avoid accidents and save drivers' lives when they become drowsy. The purpose of this paper is to create a fatigue detection system using Nvidia Jetson Nano and Python that can find sleepiness in people just by studying the live stream feed of their faces, research in this direction, but realized with a standard computer.

1877-0509 © 2024 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 11th International Conference on Information Technology and Quantitative Management
10.1016/j.procs.2024.08.101

Studies in this sense show it can be difficult to gauge someone's level of fatigue only by glancing at their face in a live stream feed; to do so would involve the application of computer vision and machine learning methods [2].
The tiredness detection system is a unique technology that analyzes a live stream feed of people's faces to detect indicators of exhaustion. This technology records the person's face using a webcam or similar equipment, which is then pre-processed to reduce any background noise and guarantee that the person's face is clearly seen. The pre-processed stream is used to extract weariness-indicating signs including drooping eyelids, closed eyes, yawning, and lip motions. These characteristics are then used to train a machine learning model to detect symptoms of tiredness in a person's face.
The most difficult part of this project is gathering the necessary data to train the machine learning model. Although there are several datasets available for training a neural network to find emotions, none of them include all indicators of weariness. The project examines if the driver's eyes are closed for more than a second as proof of concept. Finding a dataset that describes if the person is either yawning or not can prove challenging because there are a lot of human expressions with the mouth shut [1,2].
This approach proposes to run the neural network on a cheap low energy single board computer (SBC) as Nvidia Jetson Nano, almost double as a Raspberry Pi, but much more powerful, because it is powerful enough and it offers the dedicated camera port for faster transport of data to the processing unit and a GPU with cores dedicated for AI work. The Nvidia Jetson Nano is the Nvidia's response to a well-known platform, Raspberry Pi [3 4].
The Cohn-Kanade (CK) database was released in 2000 to encourage research on automatically finding the unique facial expressions.[5] The CK dataset holds seven labeled directories, each label describing a facial expression as follows: anger, contempt, disgust, fear, happiness, sadness, surprise. Despite CK dataset is one of the largest datasets with facial expressions it cannot be used for our application, but the principles of the algorithm are similar.
In other research papers [1] and [2] analyze how do appreciate when a person is tired by using standard computer system.
The algorithms and the workflow can be adapted to Nvidia Jetson, and can be accelerated because using the camera module, the information gets faster to the graphic processing unit (GPU) than by using a webcam or a USB/Wireless camera.
In earlier projects that involved machine learning Python was used successfully. The important libraries are NumPy, Scikit-learn, SciPy, Matplotlib because are used by most machine learning frameworks such as Torch and TensorFlow. For image processing the most important library is OpenCV[6-13].
The haarcascade frontalface_default.xml file is an OpenCV library pre-trained cascade classifier. It is available from the OpenCV repository or may be created by training your own cascade classifier.
Any Neural Network creates some mathematical relations between the input and the output based on a model. The usual approach for training a machine learning model is to create a labeled dataset [8, 9].
It is important to note that training a cascade classifier can be time-consuming, especially when dealing with a high amount of data. To prevent overfitting, a proper mix between positive and negative samples is also essential. As an alternative, there are pre-trained cascade classifiers in locations as OpenCV repository and CascadeClassifier that can be used directly in application [7, 12, 13].
The usual steps in this type of project are:

1. Data acquisition: capture live stream.
2. Preprocessing: remove noise.
3. Detect the region of interest: face.
4. Extract features.
5. Train a machine learning model on the labeled features.
6. Ask the model for prediction, in this case tiredness.
7. Display results. [9, 14-18]

The typical machine learning development workflow for supervised learning is represented in the picture below:

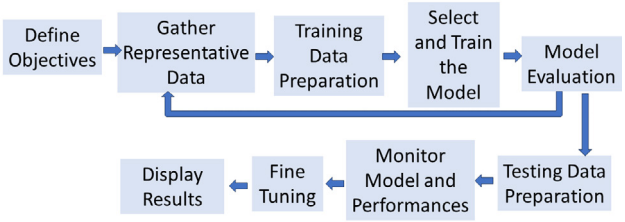

Fig. 1. Workflow in developing a Neural network model.

2. Methods and Technics

Nvidia Jetson is a series of embedded computing platforms that are designed for high-performance artificial intelligence applications. Nvidia Jetson devices have powerful GPUs and CPUs that can run complex deep learning models and algorithms with low power consumption and high efficiency. Nvidia Jetson devices can be used to build edge computing solutions that can perform tasks such as image processing, object detection, face recognition, speech recognition, natural language processing, and more.

To understand how CNNs work mathematically, we can use some equations to describe their operations. For simplicity, we will assume that the input data and filters are two-dimensional matrices, and that there is no padding or bias term. Let X be an input matrix of size m x n, and let F be a filter matrix of size p x q. Then, the convolution operation between X and F can be written as:

$$Y_{(i,j)} = \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} X_{(i+k,j+l)} F_{(k,l)} \tag{1}$$

where Y is an output matrix of size (m-p+1) x (n-q+1), and i and j are indices ranging from 0 to m-p and n-q, respectively. The convolution operation can be seen as sliding the filter over the input matrix and computing the dot product between them at each position.

The pooling operation can be seen as dividing the input matrix into non-overlapping submatrices of size r x s, and applying a function to each submatrix, such as taking the maximum or the average value. For example, if X is an input matrix of size 4 x 4, and we apply max pooling with a filter size of 2 x 2 and a stride of 2, we get an output matrix of size 2 x 2 as follows:

$$X = \begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix} \tag{2}$$

$$W = \begin{pmatrix} \max(x_{00}, x_{01}, x_{10}, x_{11}) & \max(x_{02}, x_{03}, x_{12}, x_{13}) \\ \max(x_{20}, x_{21}, x_{30}, x_{31}) & \max(x_{22}, x_{23}, x_{32}, x_{31}) \end{pmatrix} \tag{3}$$

The fully connected layer can be seen as a matrix multiplication between the input vector and the weight matrix. For example, if X is an input vector of size n x 1, and W is a weight matrix of size m x n, then the output vector Y of size m x 1 can be written as:

$$Y = W \cdot X \tag{4}$$

The activation function can be seen as applying a non-linear function to each element of the input vector or matrix. For example, if X is an input vector of size n x 1, and f is an activation function such as sigmoid or ReLU, then the output vector Y of size n x 1 can be written as:

$$Y_i = f(X_i) \tag{5}$$

where i is an index ranging from 0 to n-1.

To learn the best parameters of the CNN, such as the filter values and the weight values, we need to define a loss function that measures the difference between the predicted output and the true output. For example, if Y is the predicted output vector of size m x 1 and T is the true output vector of size m x 1 for a given input image, we can use the cross-entropy loss function as follows:

$$L = -\sum_{i=0}^{m-1} \frac{T_i}{\log(Y_i)} \quad (6)$$

where i is an index ranging from 0 to m-1. The cross-entropy loss function penalizes wrong predictions by increasing the loss value when the predicted probability is low for the true class or high for the false class.
To minimize the loss function and update the parameters of the CNN, we can use an optimization algorithm such as gradient descent or stochastic gradient descent. The gradient descent algorithm updates the parameters. The gradient descent algorithm can be written as:

$$\theta_{k+1} = \theta_k - \alpha \nabla L(\theta_k) \quad (7)$$

Where θ is a parameter vector that has all the filter values and weight values of the CNN, α is a learning rate that controls the step size of the update, and ∇L(θ) is the gradient vector of the loss function with respect to θ.
The stochastic gradient descent algorithm is a variation of the gradient descent algorithm that updates the parameters using a random subset of the training data. The stochastic gradient descent algorithm can be written as:

$$\theta_{(k+1)} = \theta_k - \alpha \nabla L(\theta_k; X^i, Y^i) \quad (8)$$

where X^i, Y^i are a random input image and its corresponding output vector from the training data. The stochastic gradient descent algorithm iterates over all the training data multiple times until convergence or until a maximum number of epochs is reached.[8]
In summary, developing a neural network involves a structured workflow that begins with collecting data, which is crucial for training the network. Once the data is gathered, the next step is to create the neural network object, followed by configuring the network's inputs and outputs. After these initial steps, the weights and biases are initialized, which sets the stage for training the network. The training process involves adjusting these parameters to minimize error and improve prediction accuracy. Finally, the network is validated and, if satisfactory, deployed for practical use.

2.1. Setting up the Nvidia Jetson Nano with a Raspberry Pi Camera Module

Begin by setting up the Nvidia Jetson Nano and installing the necessary libraries and software, including OpenCV, SciPy, NumPy and Matplotlib. Setting up the camera involves connecting the module to the Raspberry Pi board and enabling the interface [3, 5, 6, 7, 10 - 12].
Because the Raspberry Pi camera is a CSI camera, but Nvidia Jetson recognizes only MIPI cameras. Fortunately, there is a script that allows the conversion [4].
To ensure that the person's face is clearly seen, the application needs to pre-process the live stream footage to remove any noise. This may be performed with OpenCV's Haar Cascade Classifier. Faces in a picture are detected by the classifier and their coordinates are returned. Based on these coordinates, we may crop the picture and resize it to a predetermined size to ensure that all of the photos have the same dimensions. To lower the number of channels and conserve memory, we may alternatively transform the image to grayscale [1, 13, 18- 22].

2.2. Face Detection and Features Extraction

Face detection and feature extraction are critical components in the field of computer vision, playing a vital role in various applications such as surveillance systems, human-computer interaction, and identity verification. The process begins with face detection, where algorithms identify and locate human faces within images or video feeds. For this task, OpenCV has a dedicated Haar Cascade, named “haarcascade_frontalface_alt.xml”. Once a face is detected, feature extraction comes into play, which involves analyzing the facial structure to identify unique attributes like the eyes, nose, mouth, and eyebrows. These features are then used for further analysis, such as age and gender estimation, emotion recognition, biometric identification, or as in our case tiredness detection. Advancements in machine learning, particularly the development of convolutional neural networks (CNNs), have significantly improved the accuracy and efficiency of these processes. For instance, the Viola-Jones algorithm is a well-known method for face detection, utilizing Haar features and a cascaded AdaBoost classifier for rapid and reliable detection. On the other hand, deep learning architectures have been employed for real-time feature extraction, enhancing the capability to recognize and analyze facial features accurately.
The best way is to extract the region of interest (ROI) from the earlier step [23].

2.3. Training A Machine Learning Model and Using it to Predict

Train a machine learning model using a labeled dataset where each sample is labeled as "tired" or "not tired". The extracted features will be used to train the model.
Training a machine learning model to find fatigue: A machine learning model would be trained using the extracted features. A labeled dataset would be used to train the model, with each sample bearing the label "tired" or "not tired."
The machine learning model can be used to foretell weariness in the live stream feed once it has been trained. The model would evaluate the features that were extracted and forecast whether the user feels tired.
A CNN is a type of neural network that consists of multiple layers of neurons that can learn to extract features from images. A CNN can automatically learn to recognize patterns and shapes from raw pixel values without the need for manual feature engineering. A CNN usually has three types of layers: convolutional layers, pooling layers, and fully connected layers. A convolutional layer applies a set of filters to the input image to produce feature maps that highlight several aspects of the image. A pooling layer reduces the size and complexity of the feature maps by applying a down sampling operation such as max pooling or average pooling. A fully connected layer connects all the neurons from the earlier layer to the output layer that produces the final predictions.
To make a more precise model we can build with Keras a Convolutional neural network (CNN). layer, an output layer, and a hidden layer with potential for more layers. These layers are put through a convolution operation with a filter that multiplies their 2D matrices together [23].
The following layers make up the architecture of the CNN model:

- Convolutional layer with 32 nodes with kernel size 3
- Convolutional layer with 32 nodes with kernel size 3
- 64 node convolutional layer with kernel size 3
- 128 nodes; fully connected layer

The last layer has two nodes and is also completely connected. Except for the output layer, where we utilized Softmax, all the layers use a Relu activation function [23].
The architecture of the neural network is presented in figure 2:

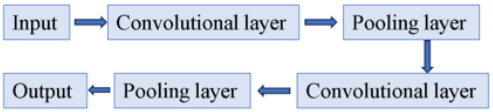

Fig. 2. Workflow of a CNN

The training parameters for a Haar cascade classifier are stored in the XML file haarcascade frontal face default.xml. A machine learning technique for detecting objects in pictures or video frames is called a Haar cascade classifier. The classifier in this instance is trained to recognize front faces in photos.
This application loads an image, converts it to grayscale, and afterwards loads the haarcascade frontal face default.xml file using the cv2.CascadeClassifier() method. The next step is to find faces in the grayscale image by using face cascade.detectMultiScale() method, then to draw rectangles around those faces using the cv2.rectangle() method. The image is then shown with the rectangles.
The edge detection model is loaded in this example using the cv2.Canny() method, and the edge detection on the grayscale frame is done using the canny.detectMultiScale() method. The final frame with detected edges is displayed after rectangles are drawn around the edges using the cv2.rectangle() technique. When the user pushes the Esc key, the remaining code ends the program and captures the live stream feed before converting it to grayscale. Here is a simple illustration of how tiredness-related features from the pre-processed data can be extracted using edge detection. You might alter this code to suit your own requirements, such as by extracting distinctive characteristics of fatigue using other computer vision techniques.
The features can be used to train a machine learning model to recognize signs of exhaustion in a person's face. We can create a new neural network or use an old one by transfer learning.
For forecasting the eye state, we are using the CNN classifier. We must conduct specific actions to input our image into the model because it requires the proper starting dimensions. First, we use r_eye = cv2.cvtColor(r_eye, cv2.COLOR_BGR2GRAY) to convert the color image to grayscale. The image is then scaled down to 24 * 24 pixels because our model was trained on 24 * 24 pixels images using the function cv2.resize(r_eye, (24,24)). With

r_eye = r_eye/255, we normalize our data for greater convergence. All values will range from 0 to 1. Boost the dimensions to add to our classifier's input.
We used model = load_model('models/cnnCat2.h5') to load our model. We now use our model, lpred = model.predict_classes(l_eye), to predict each eye. When lpred[0] = 1, it shows that the eyes are open, and when lpred[0] = 0, it shows that the eyes are closed.
An example of the training datasets can be seen in figures 3-6. [26]

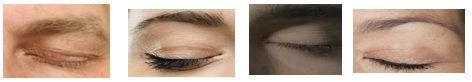

Fig. 3: Examples of Eye Closed Dataset

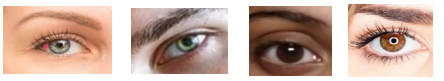

Fig. 4: Examples of Eye Opened Dataset

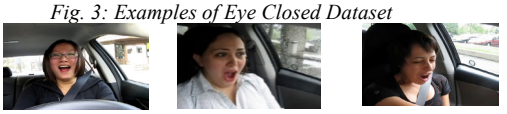

Fig. 5: Examples from the Yawn Dataset

Fig. 6: Examples from No Yawn Dataset

Transfer learning is a technique that allows us to reuse a pre-trained model for a new task by transferring its learned knowledge and features. Transfer learning can save us time and resources by avoiding training a model from scratch on a large amount of data. Transfer learning can also improve the performance of the model by using the generalization ability of the pre-trained model. Transfer learning usually involves two steps: freezing and fine-tuning. Freezing means keeping the weights of some, or all of the layers of the pre-trained model fixed and not updating them during training. Fine-tuning means updating the weights of some or all of the layers of the pre-trained model with a small learning rate to adjust them to the new task.
In our case is better to train the model on the PC, because it has more processing power, and use transfer learning to the Nvidia Jetson Nano.
After training, the machine learning model may be used to forecast weariness in the live stream feed.
The clf.predict() technique is used in this example to predict if the user is exhausted or not based on the collected information. Using the cv2.putText() function, the prediction is then shown on the frame. When the user pushes the Esc key, the software stops with the final frame displaying the prediction.
It should be noted that the extract features() method is not used in this example, and the labeled dataset is presumed to be accessible in a CSV file. To extract the features associated with weariness from the pre-processed feed, use the extract features() method and construct a labeled dataset for training the model.
The score is a number that will be used to calculate how long the subject has kept his eyes closed. Therefore, if both eyes are closed, we will continue to increase the score, and if one eye is open, we will drop the score. Using the cv2.putText() function, we are drawing the outcome on the screen to show the person's status in real time.

2.4. Displaying the Results

Display the results of the prediction on the screen. This can involve showing a message or a visual indicator, such as a bar graph or a green/red light, which shows the level of tiredness taking in consideration a simple rule-based system to find the level of tiredness of the person based on the frequency and duration of each category.
Configure the system to alert the user in case of elevated levels of tiredness, such as by playing a sound.
These are a few techniques the system might notify the user:

- Sound: To inform the user of excessive sleepiness, the device might be designed to make a certain sound, such as an alarm or a beep using the GPIO pins to connect a buzzer.
- To show the amount of weariness, the system might display a visual signal, such as a flashing light or a colored LED using the GPIO pins.

Design a graphical user interface (GUI) to display the results in a more visually appealing way and to supply more options to the user.

2.4. Fine-tuning the Model

To improve the accuracy of the model, one needs to fine-tune the system by using a larger dataset, trying out

different machine learning algorithms, or tweaking the model parameters.
Certainly, fine-tuning the machine learning model is an essential step in improving the system's accuracy. Fine-tuning means updating the weights of some or all of the layers of the pre-trained model with a small learning rate to adjust them to the new task. Fine-tuning allows us to fine-tune or refine the features learned by the pre-trained model to better suit our specific task. Fine-tuning requires less data and less training time than training from scratch, but it also requires careful tuning of the learning rate and other hyperparameters to avoid overfitting or underfitting.
Fine-tuning works by using an optimization algorithm such as stochastic gradient descent (SGD) or Adam to minimize a loss function that measures the difference between the predictions of the model and the true labels of the data. The gradients show how much each weight should change in order to reduce the loss. The learning rate decides how big or small each weight update should be. A small learning rate means smaller weight updates and slower convergence but more stability. A large learning rate means larger weight updates and faster convergence but more instability.
Here are several ways to fine-tune the model:

- Get a larger dataset. A larger dataset can help the machine learning model in learning better features and patterns in the data, improving the model's accuracy.
- Experiment with different machine learning methods. Several different machine learning algorithms, such as decision trees, random forests, and neural networks, may be used for categorization.
- Model hyperparameter tuning. Several machine learning algorithms feature hyperparameters that may be tweaked to improve model performance. random search.
- Employ data augmentation. Data augmentation is the process of producing new training samples from existing data by adding transformations such as rotations, flips, or brightness modifications. This can aid in increasing the dataset's size and improving the model's ability to generalize to new data.
- Handle class imbalance. If the dataset is unbalanced, which means there are disproportionately more samples of one class than the other, the model's performance may suffer. To alleviate class imbalance, approaches such as oversampling, under sampling, and synthetic minority oversampling technique (SMOTE) might be used.

The optimization algorithm works iteratively by repeating two steps: forward propagation and backpropagation. The chain rule allows us to compute the gradient of a composite function by multiplying the gradients of each sub-function.
The system's accuracy and reliability for recognizing weariness in live stream feeds by fine-tuning the model may increase.

3. Results

Using a Nvidia Jetson Nano and OpenCV, the suggested system detects weariness in real time. The system's accuracy may be enhanced by fine-tuning the machine learning model with a larger dataset. Also, the model can be trained on other datasets in order to learn more signs of tiredness, thus increasing the accuracy and cutting some of the false positive predictions. To make sure that all false positive predictions are eliminated, the next iteration will be able to connect to a smartwatch that checks the heart rate and blood pressure for a more exact determination. The results of the model's performance are shown in the Figures 7 and 8.

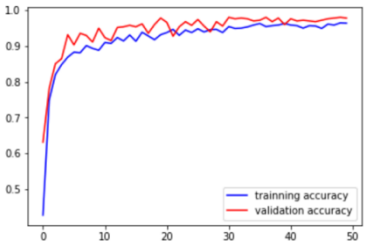

Fig 7 Accuracy Plot

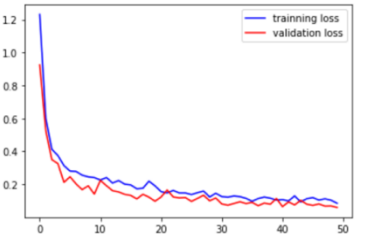

Fig 8 Loss Plot

4. Conclusions

This paper showed how to develop a weariness detection system with Nvidia Jetson Nano, Raspberry Pi Camera, and Python scripting.
The device may find exhaustion and inform the user in cases of excessive degrees of fatigue by assessing a live stream feed of a person's face and the system may be fine-tuned to increase its accuracy, and a graphical user interface can be designed to make it more user-friendly.
Using CNN there will be some changes in accuracy and performance. Because the Nvidia Jetson has a reasonable amount of memory and processing power the performance will be unnoticeable, but the accuracy will be much better. Since the algorithm will feed the ROI into the classifier it will enable the system to decide how tired a person is.
The tiredness detection system is a promising technology that can aid in the prevention of accidents caused by driver exhaustion and other fatigue-related events. It may be expanded to look at a range of data sources, including a person's facial expressions, posture, body language, speech patterns, and other relevant information, to properly find their level of weariness.
To assess the effectiveness of the proposed solution, simulation results are presented. The work confirms the efficiency of the proposed approach, extremely important in estimation of the transport security.
As perspective, a possible recommendation is to transfer the theoretical and simulation results on a real traffic network system.

References

[1] Mahek Jain, Bhavya Bhagerathi, Sowmyarani C N “Real-Time Driver Drowsiness Detection using computer Vision,” JEAT(Online), Volume-11 Issue-1, October 2021
[2] S. Pattanaik, D. K. Sahu, and R. K. Mahapatra, "Real-time Driver Fatigue Detection System using Eye Blink Analysis
[3] Raspberry Pi. (n.d.). Retrieved from https://www.raspberrypi.org/
[4] Jetson Nano Developer Kit | NVIDIA Developer
[5] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar and I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, San Francisco, CA, USA, 2010, pp. 94-101, doi: 10.1109/CVPRW.2010.5543262.
[6] Python documentation: https://www.python.org/doc/
[7] OpenCV documentation, https://docs.opencv.org/master/d9/df8/tutorial_root.html
[8] Chaabene S., Bouaziz B., Boudaya A., Hökelmann A., Ammar A., Chaari L. (2021). Convolutional Neural Network for Drowsiness
[9] Weng, J., Zhang, Y., & Lin, W. (2019). Support vector machines: theory and applications. Springer.
[10] NumPy. (n.d.). Retrieved from https://numpy.org/
[11] SciPy. (n.d.). Retrieved from https://www.scipy.org/
[12] Matplotlib. (n.d.). Retrieved from https://matplotlib.org/
[13] "OpenCV-Python Tutorials." OpenCV, opencv.org/learn/.
[14] "A Guide to Face Detection in Python." Real Python, 24 May 2021, realpython.com/face-detection-in-python-using-a-webcam/.
[15] G. Hinton, "Deep Learning - A Tutorial," 2015.
[16] R. Polikar, "The Art of Feature Engineering," IEEE Signal Processing Magazine, vol. 33, no. 1, pp. 23-22, 2016.
[17] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc.
[18] "A Beginner's Guide to Machine Learning with Scikit-Learn." DataCamp, 16 June 2021, datacamp.com/community/tutorials/machine-learning-python
[19] A. Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks," Proceedings of the 25th International Conference on Neural Information Processing Systems, pp. 1097-1105, 2012.
[20] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," Springer, 2009.
[21] Face Detection using MTCNN — a guide for face extraction with a focus on speed: https://towardsdatascience.com/face-detection-using-mtcnn-a-guide-for-face-extraction-with-a-focus-on-speed-c6d59f82d49
[22] Machine Learning for Image Processing: https://towardsdatascience.com/machine-learning-for-image-processing-3618f68278e9
[23] Driver Drowsiness Detection System with OpenCV & Keras - DataFlair (data-flair.training)
[24] Rosebrock, A. (2019). Deep Learning for Computer Vision with Python: Starter Bundle. PyImageSearch.
[25] Jetson Nano Developer Kit | NVIDIA Developer
[26] https://www.kaggle.com/datasets/serenaraju/yawn-eye-dataset-new