CALIFORNIA STATE UNIVERSITY SAN MARCOS

PROJECT SIGNATURE PAGE

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE
IN
COMPUTER SCIENCE

PROJECT TITLE: Driver Drowsiness Detection on Raspberry Pi 4

AUTHOR: Shirisha Vadlamudi

DATE OF SUCCESSFUL DEFENSE: 12/07/2020

THE PROJECT HAS BEEN ACCEPTED BY THE PROJECT COMMITTEE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE.

Ali Ahmadinia	Ali Ahmadinia	12/15/2020
PROJECT COMMITTEE CHAIR	SIGNATURE	DATE
Ahmad R. Hadaegh	[illegible]	Dec 15, 2020
PROJECT COMMITTEE MEMBER	SIGNATURE	DATE
PROJECT COMMITTEE MEMBER	SIGNATURE	DATE

California State University
SAN MARCOS
ALLIANCE TO ACCELERATE
EXCELLENCE IN EDUCATION

Driver Drowsiness Detection on Raspberry Pi 4

Shirisha Vadlamudi and Advisor: Dr. Ali Ahmadinia

Graduate student, Department of Computer Science and Information Systems,
California State University, San Marcos, CA 92096,
Vadla002@cougars.csusm.edu
Department of Computer Science

December 11, 2020

Contents

Chapter 1

Abstract

Motor vehicle crashes involving drowsy driving are huge in numbers all over the world. Many studies revealed that 10 to 30% of crashes are due to drowsy driving. Fatigue has costly effects on the safety, health, and quality of life. We can detect this drowsiness of driver using various methods, e.g., algorithms based on behavioral gestures, physiological signals and vitals. Also, few of them are vehicle based. Drowsiness of driver was detected based on steering wheel movement and lane change patterns. A pattern is derived based on slow drifting and fast corrective steering movement. We developed a prototype that detects the drowsiness of an automobile driver using artificial intelligence techniques, precisely using open-source tools like TensorFlow Lite on a Raspberry Pi development board. The TensorFlow model is trained on images captured from the video with the help of object detection using Cascade Classifier. In order to have a better accuracy, an Inception v3 architecture is used in pre-training the model with the image dataset. We create and train the final model using long short-term memory (LSTM) and then convert the final TensorFlow model to TensorFlow Lite model and use this lite model on Raspberry Pi board to detect the drowsiness of driver.

Chapter 2

Introduction

Applying the object detection algorithms in real time systems could help us resolve many problems that we currently face. Accidents due to drowsiness is one of them. National Highway Traffic Safety Administration of Unites States estimates that in 2017, 91,000 police-reported crashes involved drowsy drivers. These crashes led to an estimated 50,000 people injured and nearly 800 deaths [1]. Drowsy driving can be detected and alerted to avoid motor vehicle crashes which in turn could save hundreds of thousands of lives. Computer vision and machine learning have created an all-new ecosystem for the tech industry. This duo solves complicated problems and provides better solutions in health-sector, cyber-security, wireless communication networks, etc.

2.1 Goal

The main objective of this work is to study, document, and implement ways of drowsiness detection for automobile drivers.

2.2 Contribution

Driver drowsiness detection offers a mechanism to interpret the video of the driver in a vehicle and determine the drowsiness of driver based on behavioral gestures on a Raspberry Pi 4 device. We developed a TensorFlow Lite model using artificial intelligence techniques such as convolutional neural network (CNN) (Inception v3) & recursive neural network (RNN) (LSTM) with python open-source APIs. The developed TensorFlow Lite model is loaded on the Raspberry Pi 4 board and tested on the images captured from the video of the driver to detect the drowsiness of the driver with good accuracy. The video of the driver is captured using OpenCV module in Python and different object detection and cascade classifier techniques were applied to convert the video into multiple frames and images. The proposed approach will be evaluated in terms of accuracy and performance with its desktop version as well as literature.

Chapter 3

Related Work

Multiple approaches were used by researchers in the past to detect the drowsiness of a driver. We can classify them into the following categories:

1. Driving pattern
2. Psychological and physiological behavioral
3. Vision/eye-behavioral based

Measuring drowsiness of driver based on driving pattern is developed by monitoring the movement of steering wheel. A research in this area was conducted by Krajweski et. al. [2] and achieved an accuracy of 86.1% for drowsiness detection. The study was done in four phases. 1. Collecting driving data; 2. Feature extraction; 3. Statistical model building and 4. Testing the models on test data. The key to this is feature extraction and it was classified into 3 types. a) Time domain; b) Frequency domain; c) State space. Distances, amplitudes, entropy and mean velocity of steering angle were part of time domain. Framing of the signal and computing the power spectral density per frame fall under Frequency domain. Extracting non-linear properties of steering angle signal come under state space.

Psychological and physiological characters were captured using sensors like electroencephalogram (EEG), electrocardiogram (ECG), electromyogram (EMG), Electro-Oculogram (EOG), Percentage of Eye Closure (PERCLOS) and Brain Computer Interface (BCI)/functional near-infrared spectroscopy(fNIRS) were used to analyzed and determine the level of drowsiness. Papadelis et al. [3] have developed a drowsiness monitoring system using electrophysiological systems. Analysis was performed on the data collected in two dimensions. 1) Macroscopic analysis which estimates on-going temporal evolution of physiological measurements. 2) Microscopic analysis which focuses on the physiological measurements before, during and after driving errors. In this technique, the biggest disadvantage is to connect the sensors to the human (driver) body. This requirement leads to cumbersome design and put extra hassle to the driver.

Vision and behavioral based algorithms are the most explored research areas with regards to drowsiness monitoring. Face identification from the detected object in real time was done by Viola and Jones in [4]. This process is one of the approaches that has been widely followed to detect the drowsiness. Object detection using Haar Cascade classifier is an effective method proposed by Viola and Jones [4]. After performing face detection using Haar Cascade Classifier, researchers have applied different techniques to analyze the output of face detection. Audrey H. et. al. [5] applied eye aspect ratio (EAR) algorithm to detect the eyeblinks after the post-face detection using Haar Cascade Classifier for drowsiness detection. Mohammad H. et. al. [6] worked on the drowsy detection using Haar cascade classifier and image processing techniques. They applied image processing technique OTSU (Nobuyuki Otsu) to determine the threshold and then converted the images of eyes to binary and apply Percentage of Eye Closure (PERCLOS) to determine the drowsiness of the driver. It is found that eye closure duration and blink frequency have a direct ratio of drivers' levels of drowsiness. The mean of squares of errors for data trained by the network and data into the network for testing were 0.0623 and 0.0700, respectively. Meanwhile, the accuracy of the detecting system was 93%. Ghassan J. et. al. [7] proposed drowsiness detection using Haar cascade classifier and support vector machine technique (SVM). SVMs can effectively solve linear and non-linear classification problems. Their developed algorithm achieved 93.74% accuracy in terms of performance. Niloufar A. et. al. [8] developed a system for drowsiness detection using Haar cascade classifier and Local Binary Pattern (LBP) feature representations. LBP helps in encoding the texture features of the micro-patterns of faces efficiently. This system resulted in an accuracy of rate 96%.

3.1 Cascade Classifier

Eye Detection is the key to predict the driver's drowsiness condition. Object detection using machine learning helps us achieving faster results. Haar feature based cascade classifier is one of the effective and popular object detection method available in the current world. This object detection method was proposed by Paul

Viola and Michael Jones in 2001[4]. This is a machine learning based approach where a number of positive and negative images are provided as inputs to train a cascade function. After the training, Haar features such as edge, line, and four-rectangles are used to extract features of the image using the algorithm. The extracted feature is a single value obtained by subtracting the sum of pixels under the white rectangles from the sum of pixels under the black rectangles.

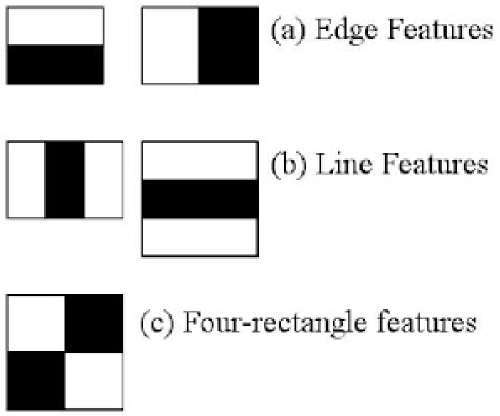

Figure 1. Haar Features [22]

3.2 CNN

Convolutional Neural Network (CNN) is a deep learning algorithm that is most commonly used for analyzing visual imagery [17]. It takes an image as an input, assigns importance (learnable weights and biases) to various aspects/objects in the image and differentiates one from the other. In a CNN network, each neuron in a layer is connected to all neurons in the next layer. CNN takes the advantage of hierarchical pattern in data and assemble more complex patterns using simpler patterns. The amount of preprocessing done in CNNs is much lower when compared to the other image classification algorithms. CNN architecture has one input and one output with multiple convolutional layers inside. Each convolutional layer processes the data and passes it on to the next layer. A CNN is able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture of CNN performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights.

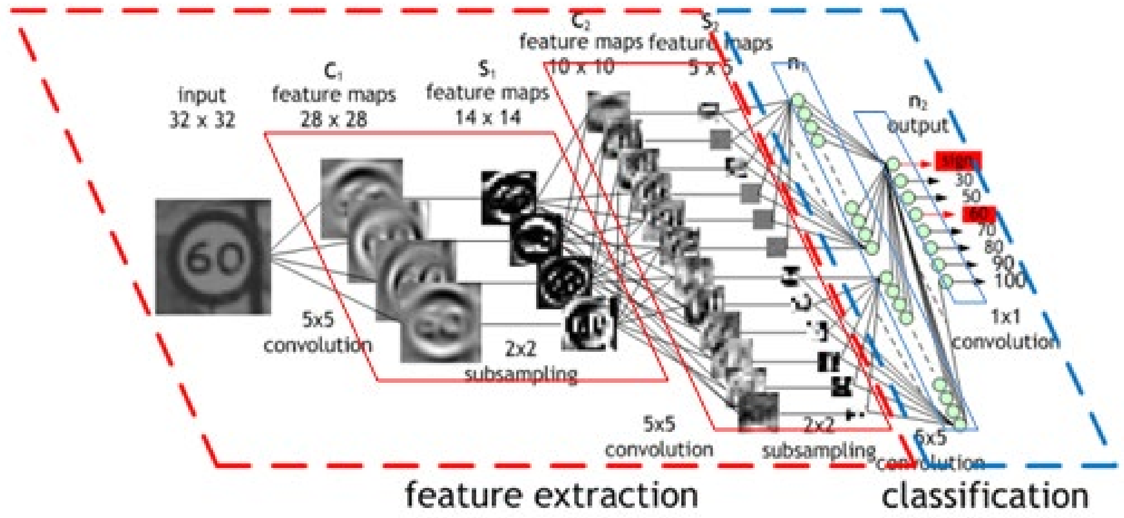

Figure 2. Feature extraction and classification – CNN [23]

Jongmin Y. et. al. [9] proposed a condition-adaptive representation for learning framework based on 3D-deep CNN, which consists of a spatio-terminal representation learning, scene condition understanding, feature fusion, and drowsiness detection. The proposed system was evaluated on NTHU data set. Rateb J. et. al. [10] used machine learning with multi-layer perceptron to detect the drowsiness. They extended their work to increase the accuracy of drowsiness detection by utilizing facial landmarks captured by camera and passed them to CNN. An average of 83% accuracy was achieved by this proposed lightweight model. Note that this method was implemented on Android smart devices. Bhargava R. et. al [11] proposed a real-time drowsiness detection based on deep learning. The proposed model achieved an accuracy of 89.5% on three level classifications. In their seminal work in [11], they proposed baseline-4, baseline-2, and compressed-2 models for drowsiness detection. However, the limitation of these models is extensive usage of memory.

3.2.1 Inception v3

Inception models are CNNs designed mainly by Google Inc. for image classification. Image classification maps the pixels of an image to linear scores (logits) for membership in the classes of a taxonomy selected by the module publisher [18]. This allows consumers to draw conclusions from the particular classification learned by the publisher module, and not just by its underlying features. This model has inception blocks that involve convolving the same input tensor with multiple filters and concatenating the filter outputs. This is completely in contrast to the other CNNs that performs single convolution operation. Inception model has evolved over the last few years and hence has multiple versions. Inception v3 is the most accurate architecture of all available architectures of CNNs for image classifications. In spite of its high accuracy, this model exhibits slower performance when compared to the MobileNet architecture.

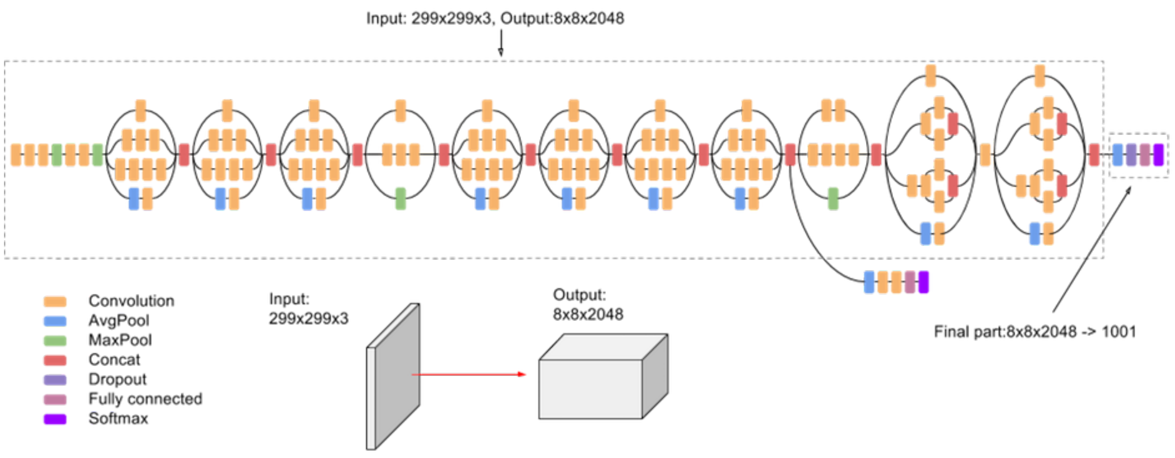

Figure 3. High level diagram of Inception v3 model [24]

Inception v3 architecture is one of the popular CNN architectures widely used in tensor flow models especially for object detection. Inception v3 was used in flower classification by Xiaoling X. et. al. in [12]. They proposed flower classification algorithm based on Inception v3 model (CNN) on TensorFlow platform, where the transfer learning technology is used to retrain the model with flower image datasets. This approach has improved the accuracy of the flower classification to a great extent. The proposed algorithm achieves 95% accuracy with diversified data sets. Adonis E. [13] conducted experiments on face shape classification using Inception-v3 architecture. The results of training accuracy and overall accuracy of Inception-v3 based face shape classification ranged from 98-100% and 84-85%, respectively when compared to the traditional classifiers whose training accuracy and overall accuracy ranged from 50-73% and 36-64% respectively. Muhammad F. et. al [14] proposed a methodology to detect drowsiness using Mobile Net CNN architecture with single shot multibox detector for object detection. Mean average precision of 0.84 is achieved with this approach.

3.3 LSTM

LSTM is a novel recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm introduced by Hochreiter and Schmidhuber [16] in 1997. LSTM is designed to overcome the error back-flow problems that exist with the RNN architectures. LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time. Traditional neural networks have the limitation of not being able to add or modify the existing information. This limitation is addressed in RNN. Moreover, LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module has a very simple structure, such as a single Hyperbolic tangent (tanh) layer [20][21]. Hyperbolic tangent (tanh) is used to determine candidate values to get added to the internal state. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four layers that interact in an efficient manner to improve the performance of conventional RNN.

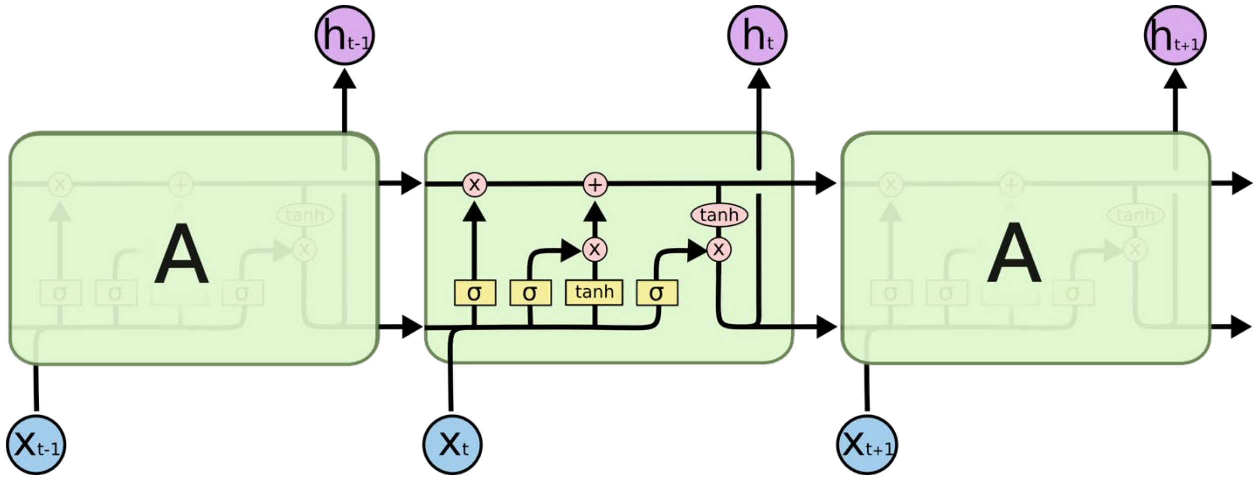

Figure 4. The repeating module in an LSTM contains four interacting layers [21].

The key to LSTMs is the cell state, which is like a conveyor belt that runs down the entire chain with linear interactions. LSTM has the ability to remove or add information to the cell state regulated by gates. Gates will let the information to go through in optional mode. LSTM has three of these gates to control and protect the cell state. The first step in LSTM is to decide on the information that needs to be discarded from the cell state. This action is then performed by the sigmoid layer, which is also referred to as "forget gate layer". Next step is to decide the new information that needs to be stored in the cell state. This modification is achieved by incorporating the input gate layer and tanh layer. Once the modification is accomplished, the cell is updated, and output is provided. Sigmoid layer decides what parts of the cell state are going to the output and we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that to decide on the output.

Ji-Hoon J. et. al. [15] proposed an approach to drowsiness detection based on deep spatio-temporal convolutional bidirectional LSTM Network (DSTCLN) using Electroencephalography (EEG) signals. In this study, the researchers classified alert, drowsy states, and five other drowsiness levels using EEG signals. Based on these characteristics, they adopted the principle of hybrid deep learning architecture (CNN + Bi-LSTM) and constructed a hierarchical CNN architecture for extracting high-level spatio-temporal features and applied the Bi-LSTM network to consider time-series data characteristics like brain signals. They achieved an accuracy level of 87%. Each of the existing proposals described above have some drawbacks. The driving pattern-based proposals do not perform well in Mountain or Hill roads. It is hard to differentiate the drowsy driving and curved road turning based on steering wheel movement. The physiological and psychological based drowsiness detection proposals are cumbersome and highly impractical as they need lot of wires and other medical equipment that needs to be attached to the human (driver) body. The drowsiness detection methods proposed using Image Processing techniques like EAR, PERCLOS, SVM and LBP techniques have limitations like usage of simulated driving environment, high configuration / high-cost devices and low accuracy. Our proposed design has the advantage of combining the best approaches for face detection, image classification techniques together to build a TensorFlow lite model which can be run-on low-cost devices.

Chapter 4

Methodology and Experiment

In this work, we focus on behavioral based algorithms using deep learning techniques. We capture the video of driver, convert the video into image frames and deploy our image classification based on deep learning models to detect the level of drowsiness and warn the users using the TensorFlow Lite model on Raspberry Pi 4 board. We used Haar Cascade classifier for object detection to convert the video into image frames which consists of left and right eye of the driver and applied Inception v3 & LSTM trained tensor flow models on the processed images to detect the drowsiness of the driver.

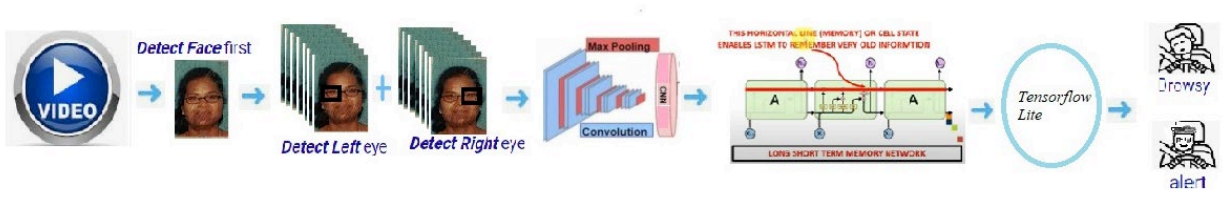

Figure 5. Sample Model

4.1 Experiment Setup

Our experiment is based on Raspberry Pi board. We capture the video of the driver using the extended camera on the Raspberry Pi board using OpenCV module of Python and then convert the video into frames and detect the eyes from the frames using Cascade Classifier.

Component	**Model**	**Configuration**
Board	Raspberry Pi 4	Ram - 4 GB CPU – Tegra Memory Card – 32 GB
Camera	Raspberry Pi Camera V2	8 Megapixel, 1080p

Table 1. Setup Configuration

4.2 Experiment Process

Figure 6 shows 7 steps process in this experiment to detect the drowsiness of a driver.

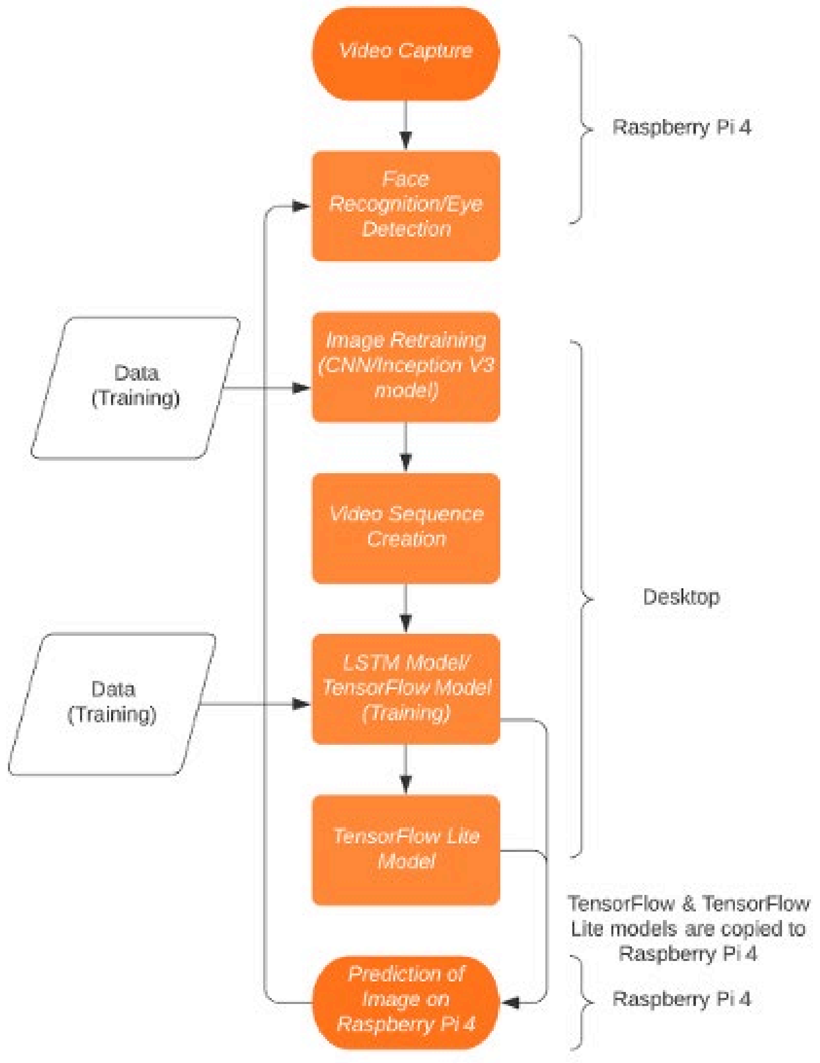
Video Capture
Face Recognition/Eye Detection
Raspberry Pi 4
Data (Training)
Image Retraining (CNN/Inception V3 model)
Video Sequence Creation
Data (Training)
LSTM Model/ TensorFlow Model (Training)
TensorFlow Lite Model
Desktop
TensorFlow & TensorFlow Lite models are copied to Raspberry Pi 4
Prediction of Image on Raspberry Pi 4
Raspberry Pi 4

Figure 6. Sequence flow diagram

Full Text: "Raspberry Pi 4" is composed of "Video Capture" leading to "face recognition/eye detection." "Desktop" is composed of "data (training): image retraining (CNN/Inception V3 model)" leading to "video sequence creation" leading to "data (training): LSTM Model/Tensor Flow Model (training)" leading to "Tensor Flow lite model." Then "TensorFlow & TensorFlow Lite models are copied to Raspberry Pi 4: "Prediction of Image on Raspberry Pi 4" which then leads back to "face recognition/eye detection".

Video Capture:

Capturing continuous facial expressions of a driver and evaluating them is the key to detect the drowsiness. In our prototype, we used Raspberry Pi 4 board and camera to capture the video of the driver. We recorded the video of the driver with 30 frames per second and a frame size of 640*480 using open cv module of python. we recorded a small video clip of one minute for offline training.

Face & Eye Detection:

Once we have our video, we detect the eyes of the driver. We first detect the frontal face of the driver and then the left and right eye of the driver using Haar-Feature based Cascade Classifier API of CV2 module in python. Once the first frame and eyes were detected, we continued to track them using Continuously Adaptive Mean-shift technique. Once we have our data/images, we used CNN to classify each frame.

Training pretrained model:

In this section, we describe the offline training and exploration using TensorFlow and Inception v3.

Image Classification is one of the toughest processes to achieve as it contains a number of parameters. Moreover, training them from scratch requires a great amount of data and computational power. Transfer learning technique can be used to reduce the amount of effort. In this step, we build a TensorFlow model by using a pre-trained and saved CNN model. We used transfer learning technique to keep the parameters for previous layers and just remove the final layer of the Inception v3 model and retrain the final layer. There are two architectures to choose from while doing this training. Inception v3 is one architecture, which gives accurate results at the expense of longer computation time. MobileNetV2 [25] is another architecture where we get the smaller and faster network but with less accurate results. However, we chose Inception v3 architecture to get the trained model with greater accuracy. The TF2 model used is trained on ImageNet [26] dataset comprising of 1000 classes for large scale visual recognition challenge. We provided training images dataset which comprised of both Alert and Drowsy classes to this model. The training image dataset comprises of 120 images each for Alert and Drowsy classes. These images were extracted from the videos we captured using Raspberry Pi 4 setup. These videos were recorded with people from both the genders and different ages varying from 30-65. All the images were without spectacles or dark shades. The name of the folders represents labels of each frame. The output of this step creates a TensorFlow model in the form of graph file. A ".pb" extension graph file is created by training the model using Inception v3 model. The pre-trained Inception v3 model we have used in our approach has 48 layers. We replace the final layer of the model with the new layer which is trained on our image dataset. So, the number of layers remain same with the new model. The number of classes in the dataset is equal to the number of output nodes in the final layer. The ImageNet [26] dataset used in our pre-trained model has 1000 classes, so the final layer has 1000 output nodes. The image dataset we have used to re-train the new final layer has 2 classes, Alert and Drowsy, so the final layer of the new model has 2 nodes.

Video Sequence Creation:

In this step, we extracted the features by passing each frame through the CNN model we created in previous step. The output which is a 2048-dimensional vector is passed to neural models and created a sequence of frames as a single vector. We created a .csv file with the information of sequences. These sequences & .csv files are provided as input to training process and hence to create the final model.

LSTM Model Training:

In this step, we created train and test data by appending the sequences created in video sequence creation process and created our final model using a sequential model using Long Short Term Memory Networks. We used single LSTM layer and added Flatten, Dense and Dropout features to it. We compiled the model using Adam optimizer with a 0.00005 learning rate.

TensorFlow Lite Model:

In this step, we converted the TensorFlow model created to TensorFlow Lite model and then transfer this lite model to Raspberry Pi 4 board. This conversion is achieved using the TensorFlow methods. We first saved the TensorFlow model signed using concrete function created using TensorSpec method and then converted the saved model using TFLiteConverter. In Table 2, we show the number of layers, neurons and the memory size of TensorFlow and TensorFlow Lite models created in our prototype. When the trained TensorFlow Model exported using Concrete function, the size of the variables is reduced to smaller size and then converted to TensorFlow Lite model. The number of layers and neurons were intact in the TensorFlow Lite model. While converting to TensorFlow Lite model it quantized the weights and converted the continuous values to a finite range of discrete values. TensorFlow Lite model could further be reduced in size using Model optimization techniques.

Table 2. TensorFlow and TensorFlow Lite model comparison

	Memory Size	**Layers**	**Neurons**
TensorFlow	941 KB (Graph) 1.06 GB (Variables)	6	3584
TensorFlow Lite	354.5 MB	6	3584

Image Prediction:

We then copied both the TensorFlow and TensorFlow Lite models to Raspberry Pi 4 and used them to predict the test data which is created from the video we captured in our first step.

Chapter 5

Results

In this section, we discuss on the numerical results that we obtained from the developed prototype. We tested both the LSTM based TensorFlow and TensorFlow Lite models on both Desktop and Raspberry Pi 4 board. We then compared their accuracy and confidence levels to show the effectiveness of our prototype. Multiple datasets are used for training and testing our models. The datasets comprise of videos shot on people without glasses, with glasses, during daytime and during nighttime. A wide variety of images were collected for the diversity.

Used Datasets:

Dataset 1: Dataset created by us.

Dataset 2: The National Tsing Hua University (NTHU) dataset used by Rateb J. et. al. [10] which comprises of 22 subjects from various ethnicities.

Dataset 3: The YawDD (Yawning while Driving) dataset used by Muhammad F. et. al [14].

Metrics:

We have trained and validated our models using the above-mentioned datasets in 3 scenarios.

Scenario-1: Training and validation were done using Dataset 1 (Our dataset).

Scenario-2: Training and validation were done using Dataset 2 (NTHU dataset).

Scenario-3: Both Dataset 2 & Dataset 3 were used. NTHU dataset is used for training and YawDD dataset is used for validation.

The accuracy and loss of the model is measured using the off the shelf method provided by TensorFlow. Table 3 shows the accuracy achieved by TensorFlow model in each scenario. For scenario-, the accuracy of the model was 75% for the 1st epoch and accuracy reached 90.91% at the end of 10th epoch. The accuracy was at 88% for the initial epoch and final accuracy was 100% after 10 epochs for both scenario-2 and scenario-3. Figure 7 shows the graph for accuracy of the model for training and validation increasing along with the number of epochs for scenario-2. Our model achieved greater accuracy when compared to the accuracy achieved by Rateb J. et. al. [10] with the same training dataset. Our model predicts the output for each input of the provided sequence and then compares expected sequence with the predicted sequence and calculates the accuracy. The formula for calculating accuracy is:

$$\text{Accuracy} = \frac{\Sigma(\text{True Positive}) + \Sigma(\text{True Negative})}{\Sigma(\text{True Positive}) + \Sigma(\text{True Negative}) + \Sigma(\text{False Positive}) + \Sigma(\text{False Negative})}$$

The Softmax layer in the LSTM model we created gives a value for each output, where all the values sum to 1. We get this value and multiply by 100 to calculate the confidence percentage of the prediction. We used TensorFlow off the shelf methods to get the values.

Table 3. Accuracy of TensorFlow model

	Accuracy at Epoch1	**Accuracy at Epoch10**
Scenario-1	75%	90.91%
Scenario -2	88%	100%
Scenario -3	86%	100%

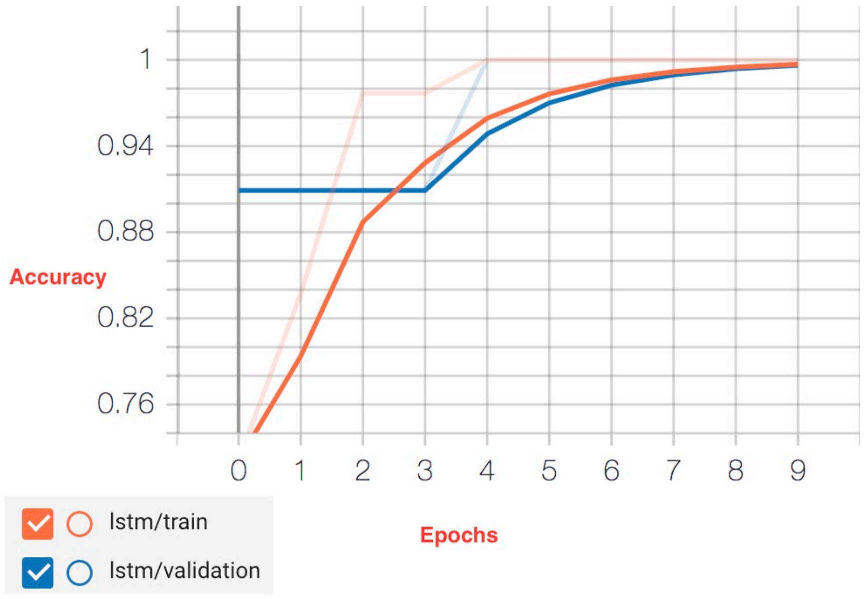

Figure 7. Tensor Board for Accuracy

Our TensorFlow Lite model predicted the alertness of the driver with a confidence of 93.61 % and drowsiness of the driver with a confidence of 92.63 % on Raspberry Pi 4. Table 4 shows the prediction time and confidence level of TensorFlow model on the Desktop and TensorFlow Lite on Raspberry Pi 4. The confidence level of the prediction on TensorFlow Lite model varies when compared to the TensorFlow prediction to a degree of seventh decimal. Table 5 show the difference in confidence levels of predictions using TensorFlow and TensorFlow Lite models. However, as expected there is a significant difference in the performance of these models on Raspberry Pi 4 when compared with the desktop version. TensorFlow & TensorFlow Lite models took an average of 0.32 seconds to predict the image on desktop whereas TensorFlow model took an average of 1.9 seconds to predict the image and TensorFlow Lite model took an average of 1.10 seconds to predict the image on Raspberry Pi 4 board. In Table 6, we compare the accuracy of our developed model with that of the model in [10].

Table 4. Confidence level and Prediction times

	Desktop			Raspberry Pi		
	Confidence Level		Time	Confidence Level		Time
	Alert	Drowsy		Alert	Drowsy	
TensorFlow	95.96%	97.90%	0.373 sec	94.50%	93.13%	1.90 sec
TensorFlow Lite	94.63%	96.50%	0.367 sec	93.61%	92.63%	1.16 sec

Table 5. Sample prediction confidence levels on Pi 4

TensorFlow Prediction on Pi 4		TensorFlow Lite Prediction on Pi 4		Prediction Confidence
Alert	Drowsy	Alert	Drowsy	
0.073621	0.926379	0.073621	0.926379	Confidence level of Validation Image 1
0.99959	0.000407	0.99959	0.000407	Confidence level of Validation Image 2
0.099091	0.900909	0.0990908	0.9009092	Confidence level of Validation Image 3
0.686709	0.31329	0.6867102	0.3132898	Confidence level of Validation Image 4

Table 6. Accuracy comparison

	Model [10]	Prototype Model
NTHU Dataset (Training and Validation)	83%	100%

All the systems proposed earlier were more or less performed on high-cost devices. Our proposed prototype makes use of TensorFlow Lite model which is designed for light weight devices and also achieves a good accuracy while using both CNN & LSTM deep learning techniques.

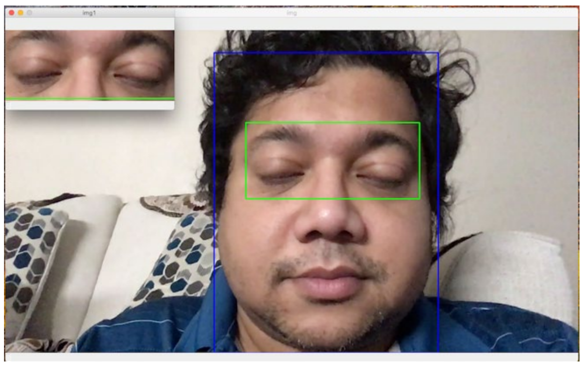
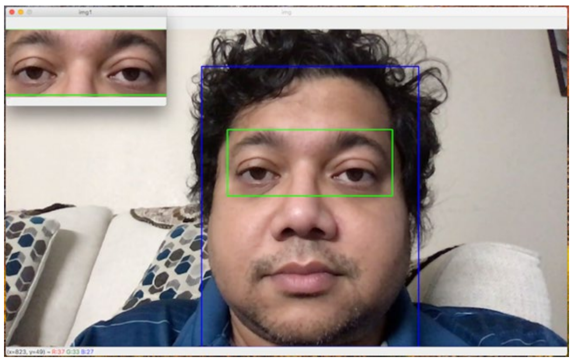

Figure 9. Validation Image 1

Figure 10. Validation Image 2

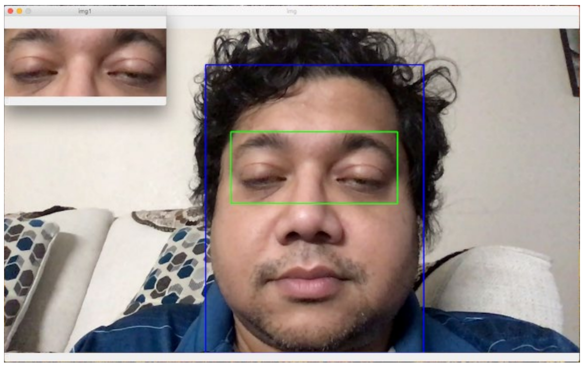
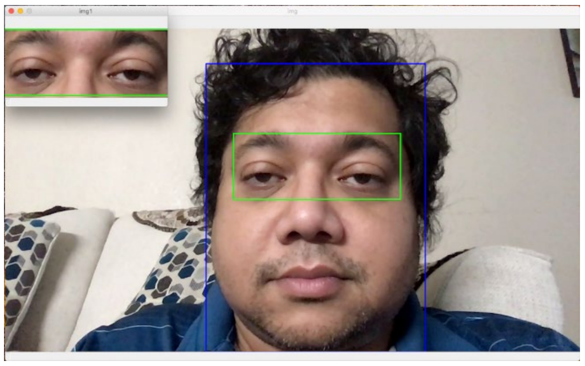

Figure 11. Validation Image 3

Figure 12. Validation Image 4

Conclusion

In this project, we developed a prototype for drowsiness detection for automobile drivers on Raspberry Pi 4. Our prototype captures video of the drivers and feed it to cascade classifier tool. Once the images are identified, they are sent to a pre-trained neural network that accurately detects the level of drowsiness of the driver. We adopted deep learning techniques, e.g., CNN (Inception v3) & LSTM to train the pattern of drowsiness from the images captured from the video. Note that the training phase is accomplished offline on a desktop. Our prototype on Raspberry Pi 4 is deployable in a wide range of automobile vehicles and can alert the drivers in a non-invasive manner while maintaining a very high level of accuracy. Raspberry Pi 4 gives the edge due to its low cost and energy efficient model.

Future Scope:

We could improve our prototype by capturing videos with various resolutions and different frame speeds. We could also try using some optimizing techniques while creating the TensorFlow Lite model and check how it impacts the prediction and accuracy.

Bibliography

[1] Drowsy Driving, https://www.nhtsa.gov/risky-driving/drowsy-driving

[2] J. Krajewski, D. Sommer, U. Trutschel, D. Edwards, and M. Golz, "Steering Wheel Behavior Based Estimation of Fatigue", *in Proceedings of the Fifth International Driving Symposium on Human Factors in Driver Assessment, Training and Vehicle Design*, pp. 118-124, 2009

[3] Papadelis et al., "Monitoring sleepiness with on-board electrophysiological recordings for preventing sleep-deprived traffic accidents", *in Clinical. Neurophysiology*, vol. 118, no. 9, pp. 1906–1922, 2007

[4] P. Viola and M. J. Jones, "Robust real-time object detection," *in International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004

[5] Nora Kamarudin1, Nur Anida Jumadi, Ng Li Mun, Ng Chun Keat, Audrey Huong Kah Ching, Wan Mahani Hafizah Wan Mahmud1, Marlia Morsin, Farhanahani Mahmud "Implementation of Haar Cascade Classifier and Eye Aspect Ratio for Driver Drowsiness Detection Using Raspberry Pi" *in Universal Journal of Electrical and Electronic Engineering* 6(5B): 67-75, 2019

[6] Mohammad Hossein Ebrahimi, Mohsen Poursadeghiyan, Adel Mazloumi, Gebraeil Nasl Saraji, Mohammad Mehdi Baneshi, Alireza Khammar "Using Image Processing in the Proposed Drowsiness Detection System Design" *in Iran J Public Health,* Vol 46, No.9, pp.1370-1377, 2018

[7] Ghassan Jasim AL-Anizy, Md. Jan Nordin and Mohammed M. Razooq "Automatic Driver Drowsiness Detection Using Haar Algorithm and Support Vector Machine Techniques" *in Asian Journal of Applied Sciences* 8(2): 149-157, 2015

[8] Niloufar Azmi, A S M Mahfujur Rahman, Shervin Shirmohammadi, Abdulmotaleb El Saddik "LBP-based Driver Fatigue Monitoring System with the Adoption of Haptic Warning Scheme" *in IEEE International Conference on*

Virtual Environments, Human-Computer Interfaces and Measurement Systems Proceedings, 10.1109, 2011

[9] J. Yu, S. Park, S. Lee and M. Jeon, "Driver Drowsiness Detection Using Condition-Adaptive Representation Learning Framework," *in IEEE Transactions on Intelligent Transportation Systems,* vol. 20, no. 11, pp. 4206-4218, Nov. 2019

[10] Rateb Jabbar, Mohammed Shinoy, Mohamed Kharbeche, Khalifa Al-Khalifa, Moez Krichen, Kamel Barkaoui "Driver Drowsiness Detection Model Using Convolutional Neural Networks Techniques for Android Application" *in IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT),* 2020

[11] B. Reddy, Y. Kim, S. Yun, C. Seo and J. Jang, "Real-Time Driver Drowsiness Detection for Embedded System Using Model Compression of Deep Neural Networks", *in 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI,* pp. 438-445, 2017

[12] Xiaoling Xia, Cui Xu, Bing Nan "Inception-v3 for Flower Classification", *in 2nd International Conference on Image, Vision and Computing,* 2017

[13] Adonis Emmanuel DC. Tio, "Face shape classification using Inception v3", *University of Philippines,* 2019

[14] M. Faique Shakeel, Nabit, A. B, Ahmad M. Anwaar, A Sohail, A Khan, and H. Rashid, "Detecting Driver Drowsiness in Real Time Through Deep Learning Based Object Detection", in *Advances in Computational Intelligence. IWANN 2019. Lecture Notes in Computer Science*, vol 11506. Springer, Cham, 2019

[15] Ji-Hoon Jeong, Baek-Woon Yu, Dae-Hyeok Lee, Seong-Whan Lee, Anam-dong, Seongbuk-ku "Classification of Drowsiness Levels Based on a Deep Spatio-Temporal Convolutional Bidirectional LSTM Network Using Electroencephalography Signals", *in Brian Sciences,* 2019

[16] Sepp Hochreiter and Jurgen Schmidhuber "LONG SHORT-TERM MEMORY", *in Neural Computation* 9(8):1735-1780, 1997

[17] Yann LeCun, Patrick Haffner, Leon Bottou and Yoshua Bengio, "Object Recognition with Gradient-Based Learning", 2004

[18] TensorFlow (https://www.tensorflow.org/hub/common_signatures/images)

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincet Vanhoucke, Andrew Rabinovich, "Going deeper with convolutions", *in IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015

[20] Blog (https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1)

[21] Blog (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

[22] Open Source Computer Vision (https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html)

[23] Blog (http://derekjanni.github.io/Easy-Neural-Nets/)

[24] Google Guides (https://cloud.google.com/tpu/docs/inception-v3-advanced)

[25] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation", CoRR, vol. abs/1801.04381, 2018

[26] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L.Fei-Fei, ImageNet (http://www.image-net.org/challenges/LSVRC/2012/) , ILSVRC-2012

Links:

https://www.python.org/downloads/release/python-370/

https://www.jetbrains.com/pycharm/

https://www.raspberrypi.org/products/

https://www.raspberrypi.org/products/camera-module-v2/?resellerType=home

Appendix:

Code Snippets:

Video capture using OpenCV library on Raspberry Pi board:

```
cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(*'MJPG')
out = cv2.VideoWriter('/home/pi/Desktop/output.avi', fourcc, 30.0,
(640,480))

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        frame = cv2.flip(frame,0)
        out.write(frame)
        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Eye detection from video:

```
# detect face in first frame
c,r,w,h = detect_one_face(frame)
pt = (0,c+w/2,r+h/2)
output.write("%d,%d,%d\n" % pt)
frameCounter = frameCounter + 1
track_window = (c,r,w,h)
roi_hist = hsv_histogram_for_window(frame, (c,r,w,h))
```

```
count = 0
term_crit = ( cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 1 )
while(1):
    ret ,frame = v.read()
    if ret == False:
        break
    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
    ret, track_window = cv2.CamShift(dst, track_window, term_crit)
    x,y,w,h = track_window
    pt = (frameCounter,x+w/2,y+h/2)
    output.write("%d,%d,%d\n" % pt)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = frame[y:y+h, x:x+w]
    eyes_left = eye_cascade_left.detectMultiScale(roi_gray)
    eyes_right = eye_cascade_right.detectMultiScale(roi_gray)
```

Re-training TensorFlow model.

```
final_tensor = tf.nn.softmax(logits, name=final_tensor_name)
tf.compat.v1.summary.histogram('activations', final_tensor)

with tf.compat.v1.name_scope('cross_entropy'):
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
      labels=tf.stop_gradient(ground_truth_input), logits=logits)
  with tf.compat.v1.name_scope('total'):
    cross_entropy_mean = tf.reduce_mean(input_tensor=cross_entropy)
tf.compat.v1.summary.scalar('cross_entropy', cross_entropy_mean)

with tf.compat.v1.name_scope('train'):
  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(FLAGS.learning_rate)
  train_step = optimizer.minimize(cross_entropy_mean)

return (train_step, cross_entropy_mean, bottleneck_input,
ground_truth_input,
        final_tensor)
```

Feature pooling:

```
def extractor(image_path):

  with open('/Users/Desktop/Siri/Project/DDD/tf_files/output_graph.pb',
'rb') as graph_file:
      graph_def = tf.compat.v1.GraphDef()
      graph_def.ParseFromString(graph_file.read())
      tf.import_graph_def(graph_def, name='')

  with tf.compat.v1.Session() as sess:
    pooling_tensor = sess.graph.get_tensor_by_name('pool_3:0')
    image_data = tf.io.gfile.GFile(image_path, 'rb').read()
    pooling_features = sess.run(pooling_tensor, \
       {'DecodeJpeg/contents:0': image_data})
    pooling_features = pooling_features[0]

  return pooling_features
```

Video sequence:

```
def extract_features():
  with open('data/data_file.csv','r') as f:
    reader = csv.reader(f)
    for videos in reader:
      path = os.path.join('data', 'sequences', videos[2] + '-' + str(26) + \
                 '-features.npy')
```

```
        path_frames = os.path.join('data', videos[0], videos[1])
        filename = videos[2]
        frames = sorted(glob.glob(os.path.join(path_frames, filename +
    '/*jpg')))

        sequence = []
        for image in frames:
          with tf.Graph().as_default():
            features = extractor(image)
            print('Appending sequence of image:',image,' of the
    video:',videos)
            sequence.append(features)

        np.save(path,sequence)
        print('Sequences saved successfully')
```

LSTM model:

```
    model = Sequential()
    model.add(LSTM(2048, return_sequences=True,
    input_shape=self.input_shape, dropout=0.2))
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(self.nb_classes, activation='softmax'))

    return model
```

TensorFlow Lite model:

```
    concrete_func = run_model.get_concrete_function(
    tf.TensorSpec([BATCH_SIZE, STEPS, INPUT_SIZE],
    new_model.inputs[0].dtype))
```

```
MODEL_DIR = "/Users/Desktop/Siri/Project/DDD/tmp/"
new_model.save(MODEL_DIR, save_format="tf",
signatures=concrete_func)
converter = tf.lite.TFLiteConverter.from_saved_model(MODEL_DIR)
lite_model = converter.convert()
```

Image Prediction:

```
# Run the model with TensorFlow Lite
interpreter = tf.lite.Interpreter(model_content=lite_model)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

expected = new_model.predict(X_test[i:i + 1])
interpreter.set_tensor(input_details[0]["index"], X_test[i:i + 1, :, :])
interpreter.invoke()
result = interpreter.get_tensor(output_details[0]["index"])
```