Fatigue Detection & Mitigation System

Peter Chen, Jorge Cotillo, Jeremey Guo, and Sanjeev Nandal
DGMD E-14, Harvard Extension School, Fall 2021
December 15th, 2021

Table of Contents

Abstract

Drowsiness conditions impacts people’s lives. In 2017 National Highway Traffic Safety Association of USA reported 795 people died from drowsy-driving and 91,000 people died from motor vehicles involving drowsy driving (Pacheco and Rehman, 2021). As COVID become a new way of life increasing number of people are working from home. Drowsiness working conditions impacts on companies’ productivities (Pedalling furiously: Workers report a 'cycle of fatigue' when working from home, 2021). This report covered two methods inspired from state-of-arts using machine learning method and the landmark extraction method and created a unique combination of the two.

The overall results showed an effective system to detect drowsiness, alert the user, and inform and monitor in real-time when user experience the drowsy conditions. The Machine learning method achieved almost 100% in validation accuracy and the landmark extraction method has been tested to effectively detect drowsiness gestures. The paper also highlights the advantages and limitations for each method and discussed how the future products may be enhanced.

Literature Review

Drowsy driving is categorized as awake, rapid eye movement, and non-rapid eye movement (Sahayadhas, Sundaraj and Murugappan, 2012). The non-rapid eye movement can be divided into three different levels: the transition from awake to sleep, light sleep, and deep sleep (Sahayadhas, Sundaraj and Murugappan, 2012). Most of the engineering controls can be categorized into vehicle-based, behavioural-based, and physiological based (Sahayadhas, Sundaraj and Murugappan, 2012). The more accurate and practical system would need to contain at least two levels of control (Sahayadhas, Sundaraj and Murugappan, 2012).

Vehicle-based monitors motion of the wheels, lanes and speed of vehicle assist for other detection system to determine if the driver is at a drowsy state and behavioural based focuses on driver’s behaviours such as eye closing, position of head and yawing (Sahayadhas, Sundaraj and Murugappan, 2012). These behaviours are typically monitors by an on-board camera connected to a single board computer (SBC). Physiological based uses EGG device to detect brain wave change when the driver is in a drowsy state (Sahayadhas, Sundaraj and Murugappan, 2012).

Adrian Rosebrock’s covered a method of drowsiness detection in behavioural based control in one of his blogs named “Drowsiness detection with OpenCV”

By having an on-board camera connected to a personal laptop or single board computer (SBC). Adrian’s method firstly uses the facial landmark from “dilib” which produces a 68 (x,y) coordination system that maps out the parts on a whole face, this can be seen on figure 2. From there the major parts on a face can be access from indexing reference e.g., left eye with [42,48] and right eye with [36, 42] (Rosebrock, 2017).

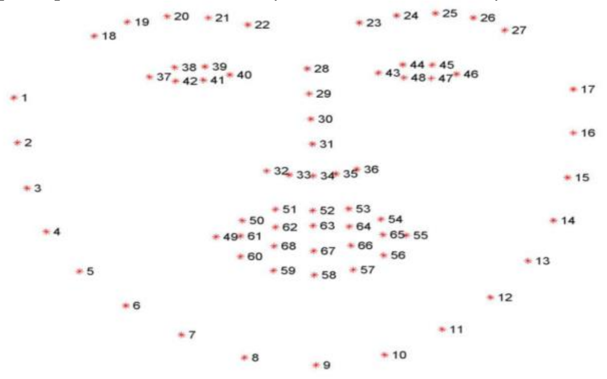


Figure 1 Facial features (Rosebrock, 2017)

In Adrian Rosebrock's another blog titled "Eye blink detection with OpenCV, Python, and dlib" (Rosebrock, 2017). Adrian covered a method which was introduced by Soukupová and Čech in their 2016 paper titled" Real-Time Eye Blink Detection Using Facial Landmarks" which uses a real time eye aspect ratio between the width of the eye the height of the eye to detect eye blinks and close this can be used to indicate driver's drowsiness level (Soukupova and Cech, 2016).
As shown in figure 2 each eye is presented by 6 (x,y) coordinates that p2-p6 and p3-p5 represents the height difference of an eye which changes as driver close/blink the eyes and p1-p4 represents the width of the eyes which stays constant regardless of the close status of the eyes.

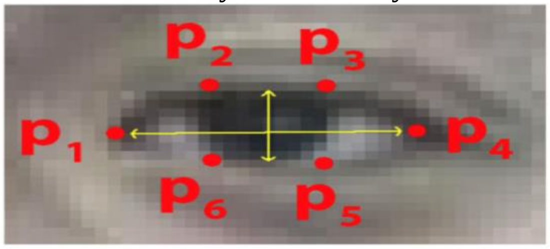


Figure 2 Eye aspect ratio (Soukupova and Cech, 2016)

Based on the work by Soukupová and Čech in their 2016 paper, "Real-Time Eye Blink Detection using Facial Landmarks".
The eye aspect ratio is calculated by:

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

Figure 3 Blink eye detection formula (Soukupova and Cech, 2016)

The denominator calculates the horizontal difference with a factor of 2 to adjust the weight in the equation as there are only one horizontal difference verse two vertical difference in the numerator. The numerator calculates the vertical difference and add them up.

From EAR, a driver with eyes open should have a fair constant EAR value while when the eyes are temporarily closed this value drops dramatically and approaching zero. The results can be seen from figure 4.

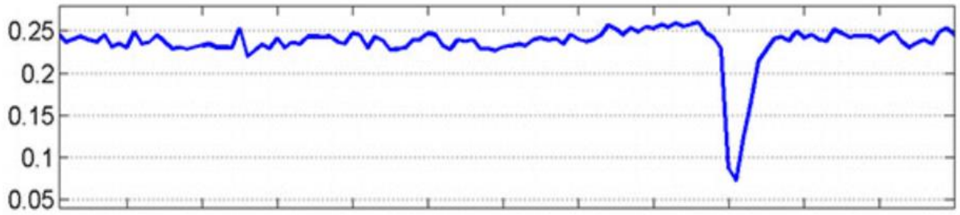


Figure 4 EAR values when eyes closed (Soukupova and Cech, 2016)

In essence from Adrian Rosebrock's covered a method using a threshold that if the eyes are closed over a certain threshold of time measured in camera frames, then it will set off the alarm otherwise the driver is deemed to be either driving with full conscious with eyes open or the driver is blinking the eyes. The method covered in Rosebrock's blog was unutilized by another group of developers to calculate the mouth aspect ratio to detect if the drivers/users are yawning (Ghosh, Bhattacharya and Saha, 2021).

Project Objectives and Scope

Project Objectives

This project aims to develop an effective system to detect drowsiness, alert drivers and enabling people to monitor the driver’s drowsiness in real-time. Ultimately to reduce the likelihood of fatality from drowsy driving and mitigating the impact to companies from drowsy working conditions.

Scope identifications

The project scopes a completion time of four weeks with a cost to build of $300 USD. This cost excludes any accessories users may need from Raspberry Pi and the packing of the product. This project assumed that user has basic knowledge to update the required libraries for the software.

Team Organization

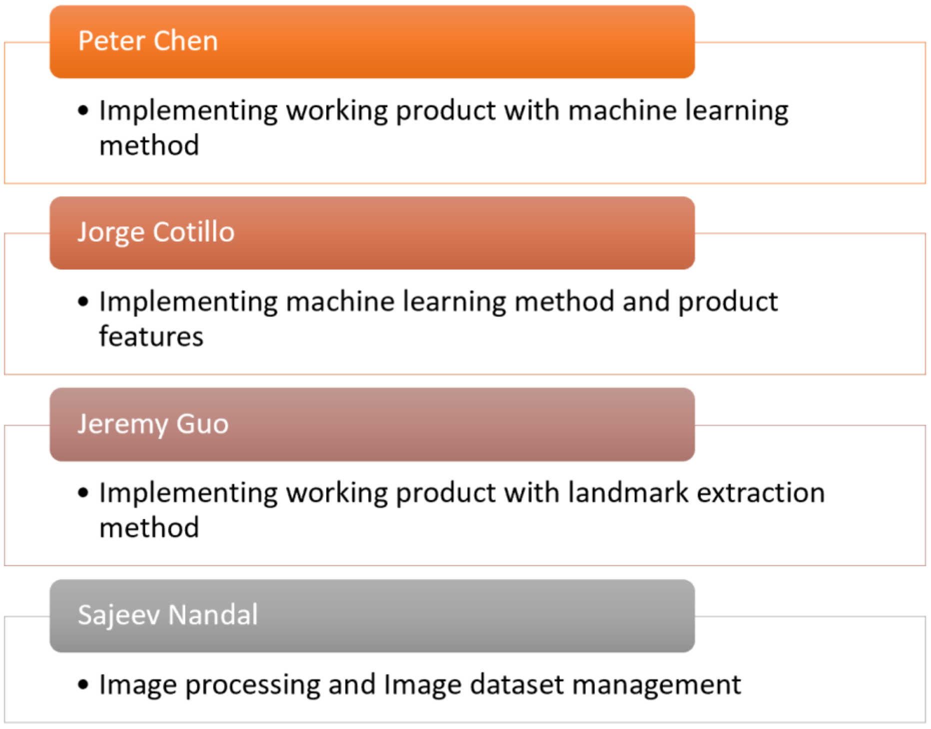

Peter Chen
• Implementing working product with machine learning method
Jorge Cotillo
• Implementing machine learning method and product features
Jeremy Guo
• Implementing working product with landmark extraction method
Sajeev Nandal
• Image processing and Image dataset management

Figure 5 Team structure and responsibility

System Architecture

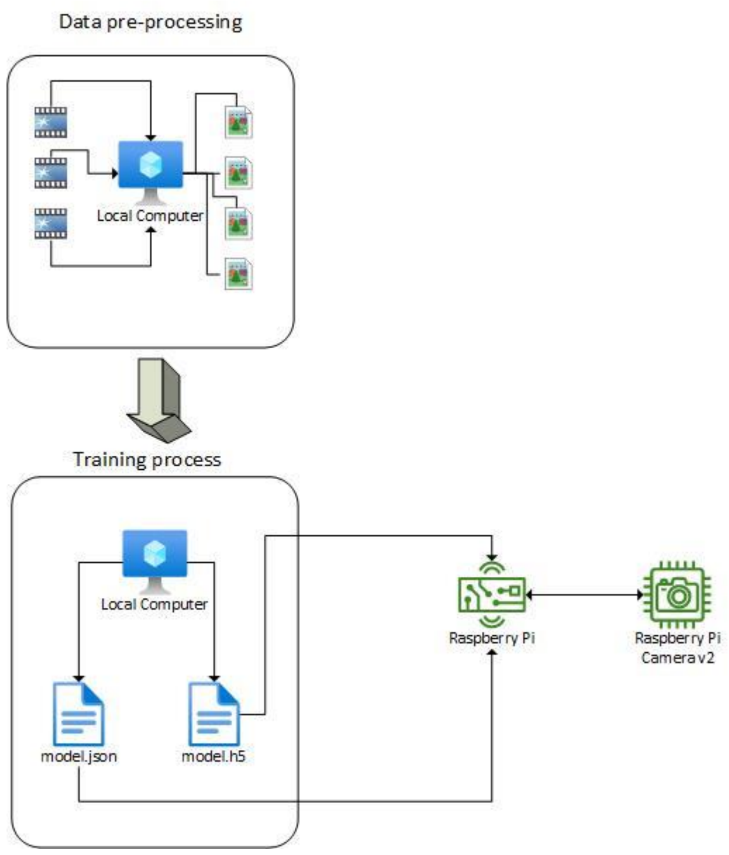


Figure 6 CNN Architecture

Fig. 6 explains the hardware architecture for CNN Model, this model requires data pre-processing, data pre-processing consists of ingesting videos – in mp4 format – capture faces (tired or alert in this specific case based on face landmarks) and saves the results as images for later training. Training process reads the saved images and labels them (tired or alert), the output of this process are two files model.json and model.h5, these two files are then imported into a Rasberry Pi to detect tired faces in real-time.

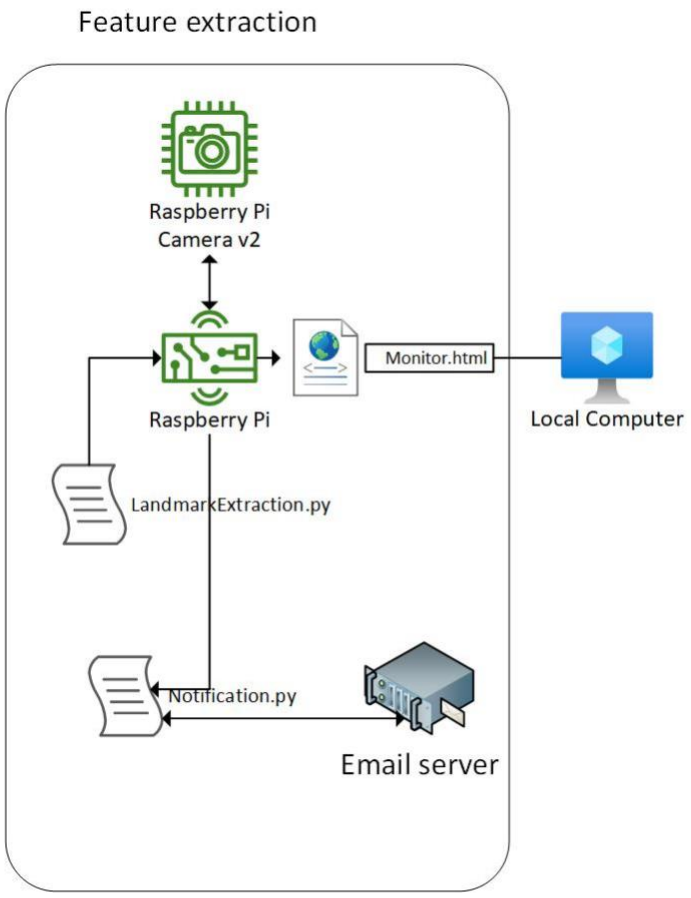


Figure 7. Feature extraction architecture

Fig. 6 shows the feature extraction process. In this case, Raspberry Pi streams real-time video by using a Raspberry Pi v2 camera, once a face is detected it executes landmark extraction, if a tired person is detected then sends a notification (sound alarm and email) to the connected user, additionally, stores the captured information and displays it in real-time via a webpage called monitor.html

Software & Packages:

Python
The project's main programming language, Python version 3.7 was used
Pandas
For data manipulation (need more info here)
Numpy
For large array manipulation. This project used numpy during data pre-processing to calculate the average of a Euclidean distance to determine if a person has her eyes closed (tired).
Tensorflow / Keras
This project uses Tensorflow / Keras to export and load a CNN model as a JSON string.
One finding is that Tensorflow / Keras is not backwards compatible.
As of this writing, Raspberry Pi uses Tensorflow / Keras version 2.2 which means loading a model generated with a version higher than version 2.2 it would not work – errors found were related to additional JSON properties added by newer versions.
Jupyter notebook
This project uses Jypyter notebook to train a CNN model. Jupyter notebook helps by showing code execution sequence, including label headings and code comments.
Recommendation is to create a requirements.txt file containing all required packages, this is an effective way to install packages prior running a notebook.
VSCode
IDE to execute Jupyter notebook, there is a VSCode plugin that helps with the execution of specific cells (or to execute all cells at once)
OpenCV / dlib
This project uses OpenCV / dlib to manipulate images (resize, change to gray scale), additionally data pre-processing uses OpenCV to capture video frames
Scikit learn
Scikit-learn is the standard machine learning library that is widely used. This project uses Scikit learn to test out machine learning models such as logistic regression, random forest, k-nearest neighbor, etc.
Mlxtend
Data pre-processing uses Mlxtend to extract face landmarks to determine if a face has her eyes or mouth closed
Anaconda
This project relies on the use of Anaconda to help with the installation of package dependencies. As an example, mlxtend requires multiple dependencies, such as joblib, pygobject, etc, Anaconda helps by installing all these dependencies on behalf of a user.

Methodology

Based on literature review, two different methods have been chosen, machine learning based using convolutional neural networks and formula-based using facial landmarks extraction. Two distinctive methods have been chosen for redundancy of the system to minimize the chances of false negatives i.e., drivers are tired but not been detected also to broaden the potential applications depends on users' needs. The facial landmarks extraction method has been chosen because its effectiveness in detecting indicative gestures and the independence of the training data. The machine learning method is chosen to detect drowsiness from the whole face rather than a localized region such as eyes or mouth.

CNN

The first approach uses a machine learning model that is often employed in computer vision called Convolutional Neural Network(CNN). Below is a stylistic and conceptual diagram of a convolutional neural network architecture. The basic idea behind a CNN is that it takes an image and converted into a matrix of numbers representing the colors. Then through each layer of the CNN, a mathematical filter is applied to extract features about the image. The first layer could extract outlines of the objects in the image. The next could potentially detect nose, eyes, ears, etc. Therefore, at each layer, the CNN goes up to next higher level of abstraction. Then at the final layer it recognizes whatever the label we assign to it. In this case, it would be tired or alert.

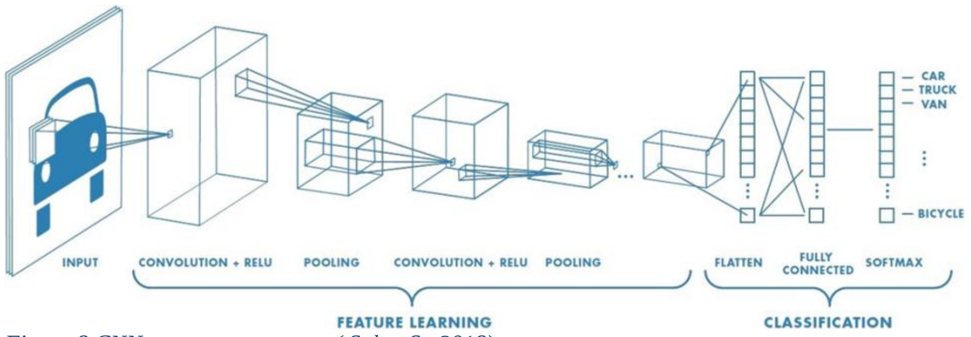


Figure 8 CNN structure courtesy (Saha, S., 2018)

Model Training & Evaluation

Facial expression using machine learning falls in a category called supervised learning. In supervised learning, training data and labels are provided and fed into machine learning algorithms. In this case, the training data are the images, and the labels are Alert and Tired emotional states. Therefore, this problem falls into the category of binary classification problem since there are two labels to classify.

Every classification machine learning model needs to be trained. The ways the image data were pre-processed for use to feed into our CNN is discussed in the image pre=processing section below.

Training & Validation:

Initially, several iterations were experimented before settling on the final training and testing setup. Initially, different categories of emotions (tired, alert, sad, angry, happy, etc.) were tested. This original setup was to see if it's possible to detect other emotions beyond tired and alert states. That experiment informed that there was insufficient training data set to detect the wide range of emotions. Additionally, the training time for this expanded set was prohibitively long (well over an hour and a half) for each run.

Additionally, original images of 48X48 pixel were used for training. However, it was discovered that this image resolution was not sufficient. Later, it was decided to expand the size of the images to 256X256 pixels. This bigger size provided better resolutions. The results below and in the appendix are based on this bigger 256X256 size images.

Finally, it was settled to limit the training data to just alert and tired images.

For training, there were:
1441 alert images
1435 tired images

For testing and validation, there were:
361 alert images
359 tired images.

Each training cycle took about an hour. The training of the CNN consists of setting how many epochs, iterations, to run. Each epoch is a learning period for the CNN. Each iteration the CNN model reduces the errors in the neural network. Thus, the more epochs the lower the errors. However, given the power of deep learning to fit data, the CNN model was able to fit both the training data and validation data with high accuracy. In fact, with just 11 epochs, the CNN model was able to achieve high 90% accuracy for both training and validation data set. Please refer to Appendix for detailed outputs for the discussion.

Please kindly note that the code was heavily inspired by the Medium's post "Facial expression detection using Machine Learning in Python"(*Analytics Vidhya*, 2021).

Transferrable Model Outputs:

Once the model is trained, the model structures need to be encapsulated for re-use. The main reason for this encapsulation of model structure is for future use without the need to retrain the model each time. This model encapsulation is done through two files:

1) model.weights: A CNN model is a multi-layer neural network. By the very nature of multi-layer neural network, it has many model weights. In fact, for this particular model, there are 36 million parameters. These weights and parameters are stored in this file.

2) model.json: JSON is one of the widely used format to store data that can be read anywhere. It's a text-based structure data format. Keras, the machine learning library that is used to do CNN modeling, allows the structure of the CNN model to be outputted in JSON format. The model structure has details such as of the number of layers, the number of activation functions, and so forth.

Data Pre-Processing

Large amount of dataset is important to get accurate models. For ML based algorithms, two types of datasets were used. First set is 111 GB of video from University of Texas Arlington (Ghoddoosian, Galib and Athitsos, 2019). This is a real-world drowsiness data collected by 60 students recording approximately 10 minutes in three different categories. Since only 36 out of 60 students allowed their faces to be used for public use, personal video stream was added to increase face counts. The second dataset was the extracted images from team personal videos. An example of images set is shown on figure 8. The steps to extract images are as follow.

- Open each video file using OpenCV.
- Run dlib based landmark feature extraction method on each frame.
- Calculate Euclidian distance using eyes dimension p2/p6 and p3/p5 as illustrated from figure 3 for eyes landmarks
- Apply a threshold to the average Euclidian distance to find frames with closed and open eyes.

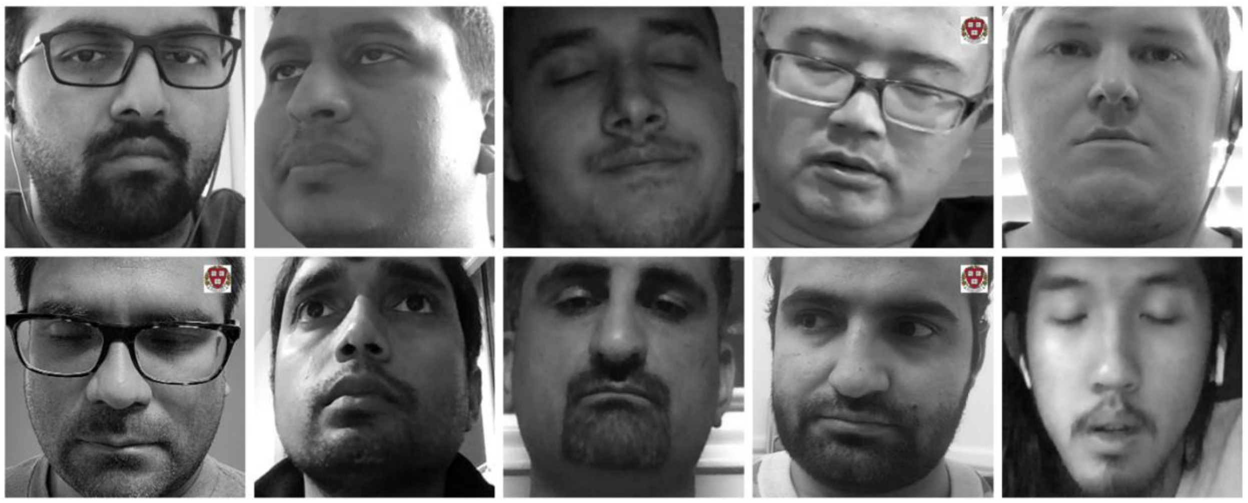


Figure 9 training data sample

Landmark extraction

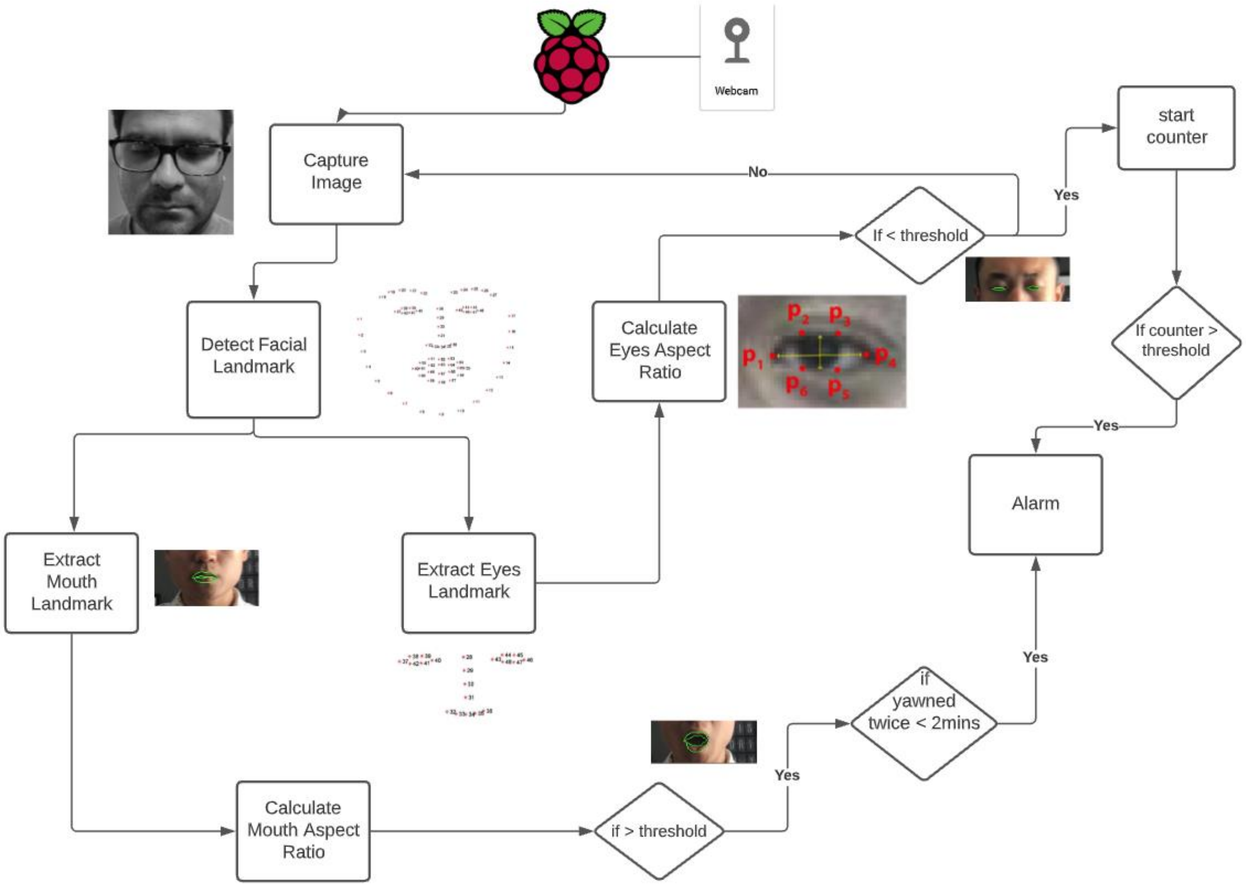


Figure 10 Landmark extraction architecture

The landmark extraction method is based on two of the most common symtoms in drowsy and fatigue conditions: yawning and micro sleep (Sunl and Rehman, 2021). The method uses the calculation covered in Soukupová and Čech's paper in 2016 titled" Real-Time Eye Blink Detection Using Facial Landmarks" to calculate the eye aspect ratio at each frame captured by the Raspberry Pi camera (Soukupova and Cech, 2016). The method followed the same structure of code with Adrian Rosebrock's blog named "Drowsiness detection with OpenCV" to detect micro-sleeps (Rosebrock, 2017). The method detects if the user/driver has closed their eyes and entered a micro-sleep state by setting a threshold for eye aspect ratio and if it falls below, it for more than a pre-defined frames/seconds then it triggers the alarm.
The landmark extract method added the method to detect if the users/drivers have their mouth open covered by state-of-art (Ghosh, Bhattacharya and Saha, 2021). Based on the number that average human yawns five to 10 times a day (Shoen and Singh, 2021). The team added a logic to detect if the user/driver has yawned twice within two minutes to sets the alarm off and wake the user up.

It is important that additional personnel can monitor user/driver's drowsiness level in real-time and be notified by emails if danger exists. PubNub eon-chart is an open-source chart and map framework for real-time data and graphing. This enabled the team to build a webpage to allow additional personnel such as employers or managers to view their employee's drowsiness status in real-time. The webpage constantly listen/subscribe to the PubNub channel and graphing the new incoming data points while the Raspberry Pi records the drowsiness level and publish it to the same channel every hour.

The drowsiness level is calculated based on each frame of micro-sleep scores three and each yawn scores one this data accumulates and get published and cleared out to the PubNub channel every hour. The team has also developed a feature to enable the additional personnel to be notified through email once the drowsiness level reach to certain level.

Results

CNN Results:

The CNN results for both training and validation are nearly perfect. However, this does not mean that it will detect alert and tired states with perfect accuracy (accuracy stats). In fact, in real-life running of the facial detection system using CNN, the results were quite mixed. Sometimes it would detect the tired state readily but at other times it's a hit and miss. The accuracy in real-time usage is quite different from the accuracy in training and in testing and validation stages.

Pros of CNN:

1) One of the major pros of using CNN is the low requirement for features engineering. Unlike other machine learning methods, CNN does not require much features engineering.
2) The second pro of CNN is that it's relatively manageable to add, remove, and change the neural network architecture. One just need to add, remove, or change a few lines of code in Tensorflow/Keras to make the necessary additions, removals, and changes. This is because each layer
3) Lastly, another strong advantage of CNN is that it recognizes the entire face holistically rather in pieces like the landmark extraction approach. This wholistic facial recognition can be used more broadly than just narrow application.

Cons of CNN:

1) The biggest drawback of CNN, like any deep learning methods, is that it requires enormous amount of data to properly train the neural network. We had about a little over a thousand images for both alert and tired.
2) The second drawback of CNN is the long training time and computational power required. We used our desktop machine to train the neural network. Even on our relatively small training data sets, it took anywhere from 45 minutes to an hour. We sometimes adjust the training data set sizes for testing purposes.
3) Like any machine learning algorithm, CNN can only learn what it has been trained on. If our training set has a limited range of types of faces, then it will only learn those types of faces. If someone came into the picture dressing like a clown and there were no clown images in the training dataset, then obviously the CNN will do a poor job recognizing a tired clown from an alert clown.

Results for landmark

The results of landmark extraction system have shown a high effectiveness in detecting two of the most common symptoms in drowsy conditions: micro-sleep and yawning. This making the landmark extraction a reliable and high ease of implementation detection system. However, it faces some limitations too.

Pros of landmark extraction

- Ease of implementation

No time and effort are required to train the system prior to use

Not depend on user to be in the training set

As it's not a machine learning based system it does not require the user to be in the training dataset for it to provide reliable detections on the indictive gestures.

- Lower computational power required

As there are no multiple layers of computation involved to decide an output, the computational power required from the SBC reduces significantly

- High reliability in detecting eyes and mouth movements

The fundamental reliability of detecting the symptoms come down to the reliability of the library of "dlib" and "OpenCV" in recognizing the whole face contour. As the two libraries have been well supported and maintained from the authors and community this drive a high accuracy in detecting the drowsiness symptoms

Cons of landmark extraction

Not a holistic detection system

As the system do not consider all regions on the user/driver's face it only looks at localized regions. It imposes a limitation that the users may act as normal but under influence from drowsiness and not been detected.

Not able pick-up sunglasses or if user cover the mouth

The system is not able to detect the eyes movement if user wear/hide their eyes movement behind the sunglasses. As the sunglasses are shaded the eyes movement are not detectable by this system. Similarly, the user can hide their tiredness by covering their mouth while yawning, which again makes the system not able to detect the mouth movement.

Future Work

1

Different levels of controls

2

Leverage cloud computing power to train models

3

Extract more features to improve drowsiness detection accuracy

4

Implement different Machine Learning models to improve accuracy

Figure11 11 Future opportunities

Future improvements can be divided into several areas
1) Different levels of controls
As identified in the literature review a truly reliable detection systems needs to include at least two level of engineering controls from: vehicle-based, behavioral-based, and physiological based. Although this report covers two distinctive methods however, they both categorize as behavioral-based engineering control hence if user hid or act in their behavior such as covering their mouth or acting on facial expressions as normal both systems will struggle to detect drowsiness or fatigue.

A common combination is using a behavioral-based system together with a psychological-based system.
From article titled "Convolutional Neural Network for Drowsiness Detection Using EEG Signals" the authors used a physiological based approach by using an EEG device to detect the state of a brain (Chaabene et al., 2021).
The authors were able to use wearable EEG device to detect the features of the drivers and feed into a CNN model to detect if they are in a drowsy state. The result suggest that the brain waves have distinctive features between a drowsy and non-drowsy state the result can be seen in figure x

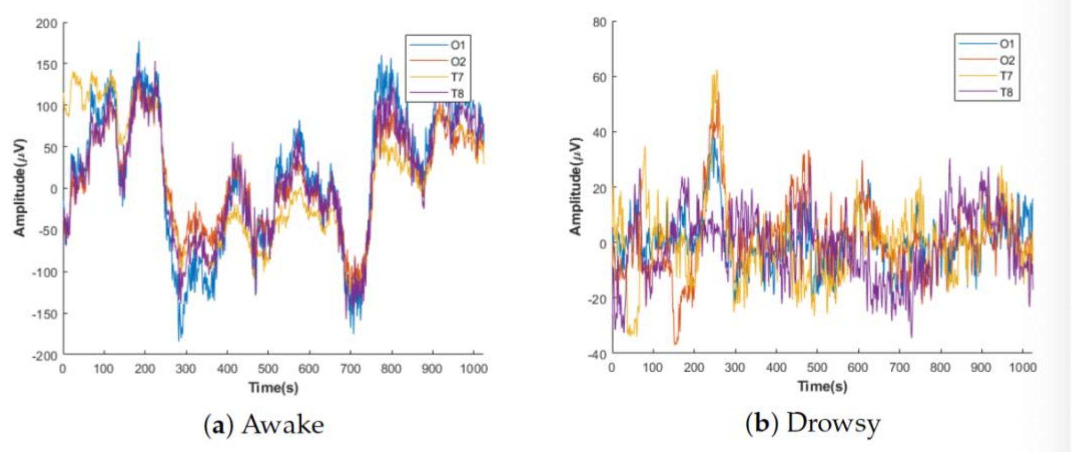


Figure12 12 Brainwave features between Awake and Drowsy (Chaabene et al., 2021)

However due to the complexity of hardware and time constraint of the project the method has not been pursuit by this project. It forms the basis for future scholars who are required to create a high accuracy detection system to pursuit further.

2) Leveraging cloud power

One of the major roadblocks for training the CNN model is the limited computation power from developer's laptop or SBC. Utilising the cloud power will significantly improve on that.

3) Extract more features

Although, yawning and micro sleep are two of the most indicative gestures. The accuracy/reliability of the system can be further improved if developer choose to include more features such as pupil changes between drowsy and non-drowsy states as Yoss, Moyer and Hollenhorst covered in their neurology paper (Yoss, Moyer and Hollenhorst, 1970).

4) Implement different machine learning models

Different machine learning models other than CNN can be further explored for improvements on the accuracy of the outputs.

Resource

Resource	Address
GitHub Repo	*jorgecotillo/DGMDE14_Final_Project_2021 (github.com)*
Demo video	*https://1drv.ms/u/s!Aqk-ltMQf5ham0i7DIvMa91Z6RsW?e=zgMJPT*
Project report	*DGMDE14_Final_Project_2021/final_report at master · jorgecotillo/DGMDE14_Final_Project_2021 (github.com)*

Cost to build

Material	Cost	Link to purchase
Raspberry Pi 4 Kit	$140	*Vilros Raspberry Pi 4 Complete Starter Kit – Vilros.com*
Raspberry Pi Camera	$25	*Official Raspberry Pi Camera Module V2 | vilros.com – Vilros.com*

Conclusion

In conclusion, drowsy and fatigue conditions impact people’s lives and company’s productivities. The team has designed a minimal viable product effectively reduce and alert drowsiness conditions. In the long run if the product can be produced at a commercial scale this will contribute towards lowering the fatality down and improving the productivities of companies up from fatigue conditions. The project has covered two distinctive methods using a CNN based method and landmark extractions method. Both methods have strength in different areas and achieved a reasonable accuracy with CNN achieved closed to 100% validation accuracy and landmark extraction can detect drowsy gestures effectively. However, they all face different limitations such as CNN requires large training data and landmark does not consider the facial expression holistically.

A more reliable detection system requires at least two levels of engineering controls from: vehicle based, behavioral based and psychological based. Due to the time constraint, this project only focused on behavioral based controls. However, all findings and future improvement opportunities have been highlighted in this document for future developer/scholar to further explore. Overall, the team has delivered an effective detection system within the budget and timeframe defined in the scope.

References

Analytics Vidhya, 2021. Facial expression detection using Machine Learning in Python. Available at: <https://medium.com/analytics-vidhya/facial-expression-detection-using-machine-learning-in-python-c6a188ac765f> [Accessed 19 December 2021].

Chaabene, S., Bouaziz, B., Boudaya, A., Hökelmann, A., Ammar, A. and Chaari, L., 2021. Convolutional Neural Network for Drowsiness Detection Using EEG Signals. *Sensors*, 21(5), p.1734.

Detecting Driver Drowsiness Based on Sensors: A Review
Ghosh, S., Bhattacharya, S. and Saha, R., 2021. *Drowsiness-Detection*. [online] GitHub. Available at: <https://github.com/fear-the-lord/Drowsiness-Detection/blob/master/README.md> [Accessed 18 December 2021].

Ghoddoosian, R., Galib, M. and Athitsos, V., 2019. *A Realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1904.07312> [Accessed 18 December 2021].

Ghoddoosian, R., 2020. *Index of /~athitsos/projects/drowsiness*. [online] Vlm1.uta.edu. Available at: <http://vlm1.uta.edu/~athitsos/projects/drowsiness/> [Accessed 19 December 2021].
Pacheco, D. and Rehman, A., 2021. *Driving While Drowsy Can Be as Dangerous As Driving While Drun| Sleep Foundation*. [online] Sleepfoundation.org. Available at: <https://www.sleepfoundation.org/drowsy-driving/drowsy-driving-vs-drunk-driving#:~:text=According%20to%20reports%20from%20the,consistent%20year%2Don%2Dyear.> [Accessed 17 December 2021].

PubNub. n.d. *Open Source Project EON - Real-time Dashboards*. [online] Available at: <https://www.pubnub.com/developers/eon/> [Accessed 18 December 2021].

Rosebrock, A., 2017. *Detect eyes, nose, lips, and jaw with dlib, OpenCV, and Python - PyImageSearch*. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/> [Accessed 17 December 2021].

Rosebrock, A., 2017. *Eye blink detection with OpenCV, Python, and dlib - PyImageSearch*. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib> [Accessed 17 December 2021].

Rosebrock, A., 2017. *Drowsiness detection with OpenCV - PyImageSearch*. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/> [Accessed 19 December 2021].

Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way*. [online] Medium. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Accessed 19 December 2021].
Shoen, S. and Singh, A., 2021. *Causes of Excessive Yawning | Sleep Foundation*. [online] Sleepfoundation.org. Available at: <https://www.sleepfoundation.org/physical-health/excessive-yawning#:~:text=On%20average%2C%20humans%20yawn%20five,times%20in%20a%20day9.> [Accessed 18 December 2021].

Soukupova, T. and Cech, J., 2016. Real-Time Eye Blink Detection using Facial Landmarks.
Sunl, E. and Rehman, A., 2021. *Drowsy Driving: Dangers and How To Avoid It | Sleep Foundation*. [online] Sleepfoundation.org. Available at: <https://www.sleepfoundation.org/drowsy-driving> [Accessed 18 December 2021].

Work in Mind. 2021. *Pedalling furiously: Workers report a 'cycle of fatigue' when working from home*. [online] Available at: <https://workinmind.org/2021/02/10/pedalling-furiously-workers-report-a-cycle-of-fatigue-when-working-from-home/> [Accessed 17 December 2021].
Yoss, R., Moyer, N. and Hollenhorst, R., 1970. Pupil size and spontaneous pupillary waves associated with alertness, drowsiness, and sleep. *Neurology*, 20(6), pp.545-545.

Zhong, G., 2019. *Drowsiness Detection with Machine Learning*. [online] Medium. Available at: <https://towardsdatascience.com/drowsiness-detection-with-machine-learning-765a16ca208a> [Accessed 18 December 2021].

Appendix

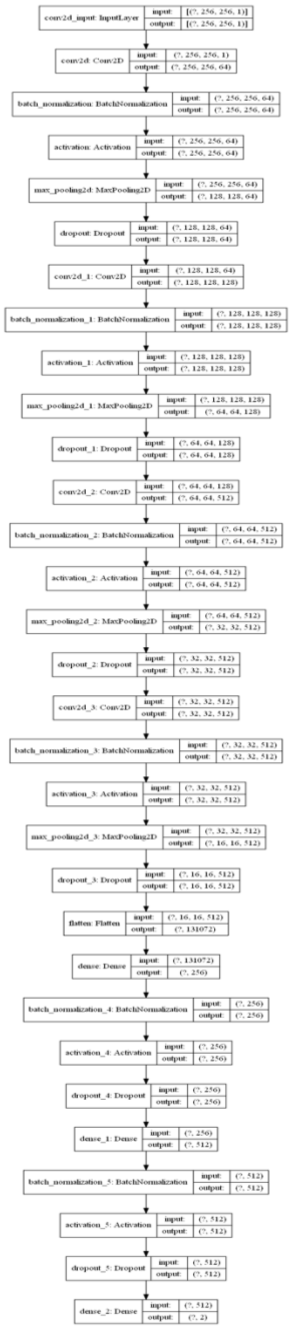
Output of CNN Model Training and Evalution:

The below shows a partial output of CNN training from epoch 11 to 15. Additionally, it also shows validation results. We can see accuracy approaching 100% rather rapidly.

```
Epoch 1/15
11/11 [==============================] - 206s 19s/step - loss: 0.1742 - accuracy:
0.9162 - val_loss: 2.0622 - val_accuracy: 0.8366
Epoch 2/15
11/11 [==============================] - 207s 19s/step - loss: 0.0299 - accuracy:
0.9924 - val_loss: 4.1166 - val_accuracy: 0.7159
Epoch 3/1511/11
[==============================] - 204s 19s/step - loss: 0.0140 - accuracy: 0.9985
- val_loss: 9.9437 - val_accuracy: 0.5185
Epoch 4/1511/11
[==============================] - 211s 19s/step - loss: 0.0062 - accuracy: 1.0000
- val_loss: 2.7871 - val_accuracy: 0.6463
Epoch 5/1511/11
[==============================] - 221s 20s/step - loss: 0.0075 - accuracy: 0.9985
- val_loss: 0.1692 - val_accuracy: 0.9403
Epoch 6/1511/11
[==============================] - 225s 20s/step - loss: 0.0068 - accuracy: 0.9985
- val_loss: 0.2740 - val_accuracy: 0.9219
Epoch 7/1511/11
[==============================] - 222s 20s/step - loss: 0.0018 - accuracy: 1.0000
- val_loss: 0.3825 - val_accuracy: 0.9048
Epoch 8/1511/11
[==============================] - 206s 19s/step - loss: 0.0037 - accuracy: 0.9985
- val_loss: 0.2838 - val_accuracy: 0.9190
Epoch 9/1511/11
[==============================] - 205s 19s/step - loss: 0.0032 - accuracy: 0.9985
- val_loss: 0.1127 - val_accuracy: 0.9361
Epoch 10/1511/11
[==============================] - 204s 19s/step - loss: 8.0350e-04 - accuracy:
1.0000 - val_loss: 0.0264 - val_accuracy: 0.9943
Epoch 11/1511/11
[==============================] - 205s 19s/step - loss: 5.5923e-04 - accuracy:
1.0000 - val_loss: 0.0063 - val_accuracy: 1.0000
Epoch 12/1511/11
[==============================] - 204s 19s/step - loss: 0.0014 - accuracy: 1.0000
- val_loss: 0.0063 - val_accuracy: 1.0000
Epoch 13/1511/11
[==============================] - 204s 19s/step - loss: 0.0014 - accuracy: 1.0000
- val_loss: 0.0115 - val_accuracy: 0.9972
Epoch 14/1511/11
[==============================] - 215s 20s/step - loss: 6.7212e-04 - accuracy:
1.0000 - val_loss: 0.0102 - val_accuracy: 1.0000
Epoch 15/1511/11
[==============================] - 215s 20s/step - loss: 3.1218e-04 - accuracy:
1.0000 - val_loss: 0.0250 - val_accuracy: 0.9972
Wall time: 57min 11s
```

Actual CNN Architecture:

Below is a visual representation of the CNN architecture.


conv2d_input: InputLayer | input: [(?, 256, 256, 1)] | output: [(?, 256, 256, 1)]
conv2d: Conv2D | input: (?, 256, 256, 1) | output: (?, 256, 256, 64)
batch_normalization: BatchNormalization | input: (?, 256, 256, 64) | output: (?, 256, 256, 64)
activation: Activation | input: (?, 256, 256, 64) | output: (?, 256, 256, 64)
max_pooling2d: MaxPooling2D | input: (?, 256, 256, 64) | output: (?, 128, 128, 64)
dropout: Dropout | input: (?, 128, 128, 64) | output: (?, 128, 128, 64)
conv2d_1: Conv2D | input: (?, 128, 128, 64) | output: (?, 128, 128, 128)
batch_normalization_1: BatchNormalization | input: (?, 128, 128, 128) | output: (?, 128, 128, 128)
activation_1: Activation | input: (?, 128, 128, 128) | output: (?, 128, 128, 128)
max_pooling2d_1: MaxPooling2D | input: (?, 128, 128, 128) | output: (?, 64, 64, 128)
dropout_1: Dropout | input: (?, 64, 64, 128) | output: (?, 64, 64, 128)
conv2d_2: Conv2D | input: (?, 64, 64, 128) | output: (?, 64, 64, 512)
batch_normalization_2: BatchNormalization | input: (?, 64, 64, 512) | output: (?, 64, 64, 512)
activation_2: Activation | input: (?, 64, 64, 512) | output: (?, 64, 64, 512)
max_pooling2d_2: MaxPooling2D | input: (?, 64, 64, 512) | output: (?, 32, 32, 512)
dropout_2: Dropout | input: (?, 32, 32, 512) | output: (?, 32, 32, 512)
conv2d_3: Conv2D | input: (?, 32, 32, 512) | output: (?, 32, 32, 512)
batch_normalization_3: BatchNormalization | input: (?, 32, 32, 512) | output: (?, 32, 32, 512)
activation_3: Activation | input: (?, 32, 32, 512) | output: (?, 32, 32, 512)
max_pooling2d_3: MaxPooling2D | input: (?, 32, 32, 512) | output: (?, 16, 16, 512)
dropout_3: Dropout | input: (?, 16, 16, 512) | output: (?, 16, 16, 512)
flatten: Flatten | input: (?, 16, 16, 512) | output: (?, 131072)
dense: Dense | input: (?, 131072) | output: (?, 256)
batch_normalization_4: BatchNormalization | input: (?, 256) | output: (?, 256)
activation_4: Activation | input: (?, 256) | output: (?, 256)
dropout_4: Dropout | input: (?, 256) | output: (?, 256)
dense_1: Dense | input: (?, 256) | output: (?, 512)
batch_normalization_5: BatchNormalization | input: (?, 512) | output: (?, 512)
activation_5: Activation | input: (?, 512) | output: (?, 512)
dropout_5: Dropout | input: (?, 512) | output: (?, 512)
dense_2: Dense | input: (?, 512) | output: (?, 2)

Below is a written-out representation of the CNN architecture:

```
Model:
"sequential"_________________________________________________________________Layer
(type)                 Output Shape              Param #
=================================================================conv2d (Conv2D)
(None, 256, 256, 64)      640
_________________________________________________________________batch_normalizatio
n (BatchNo (None, 256, 256, 64)      256
_________________________________________________________________activation
(Activation)      (None, 256, 256, 64)      0
_________________________________________________________________max_pooling2d
(MaxPooling2D) (None, 128, 128, 64)      0
_________________________________________________________________dropout (Dropout)
(None, 128, 128, 64)      0
_________________________________________________________________conv2d_1 (Conv2D)
(None, 128, 128, 128)     204928
_________________________________________________________________batch_normalizatio
n_1 (Batch (None, 128, 128, 128)     512
_________________________________________________________________activation_1
(Activation)    (None, 128, 128, 128)     0
_________________________________________________________________max_pooling2d_1
(MaxPooling2 (None, 64, 64, 128)       0
_________________________________________________________________dropout_1
(Dropout)          (None, 64, 64, 128)       0
_________________________________________________________________conv2d_2 (Conv2D)
(None, 64, 64, 512)       590336
_________________________________________________________________batch_normalizatio
n_2 (Batch (None, 64, 64, 512)       2048
_________________________________________________________________activation_2
(Activation)    (None, 64, 64, 512)       0
_________________________________________________________________max_pooling2d_2
(MaxPooling2 (None, 32, 32, 512)       0
_________________________________________________________________dropout_2
(Dropout)          (None, 32, 32, 512)       0
_________________________________________________________________conv2d_3 (Conv2D)
(None, 32, 32, 512)       2359808
_________________________________________________________________batch_normalizatio
n_3 (Batch (None, 32, 32, 512)       2048
_________________________________________________________________activation_3
(Activation)    (None, 32, 32, 512)       0
_________________________________________________________________max_pooling2d_3
(MaxPooling2 (None, 16, 16, 512)       0
_________________________________________________________________dropout_3
(Dropout)          (None, 16, 16, 512)       0
_________________________________________________________________flatten (Flatten)
(None, 131072)            0
_________________________________________________________________dense (Dense)
(None, 256)               33554688
_________________________________________________________________batch_normalizatio
n_4 (Batch (None, 256)              1024
_________________________________________________________________activation_4
(Activation)    (None, 256)              0
```

```
_________________________________________________________________dropout_4
(Dropout)          (None, 256)               0
_________________________________________________________________dense_1 (Dense)
(None, 512)              131584
_________________________________________________________________batch_normalizatio
n_5 (Batch (None, 512)              2048
_________________________________________________________________activation_5
(Activation)    (None, 512)              0
_________________________________________________________________dropout_5
(Dropout)          (None, 512)               0
_________________________________________________________________dense_2 (Dense)
(None, 2)                1026
=================================================================Total params:
36,850,946
Trainable params: 36,846,978
Non-trainable params: 3,968
```