TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Harish Kumar Singh 177319IVEM

EVALUATION OF DRIVER STATUS ASSESSMENT SYSTEM BASED ON DEEP LEARNING

Master's Thesis

Supervisor: Alar Kuusik
PhD
Supervisor: Raul Bachmann
Stoneridge Electronics Pvt Ltd.

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Harish Kumar Singh 177319IVEM

Süvaõppel põhineva sõidukijuhi seisundi jälgimissüsteemi analüüs

Magistritöö

Juhendaja: Alar Kuusik
Doktorikraad
Juhendaja: Raul Bachmann
Stoneridge Electronics Pvt Ltd.

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to this thesis has not been presented for examination anywhere else.

Author: Harish Kumar Singh

11.05.2020

Abstract

The objective of the thesis is to evaluate the deep learning approach for driver status assessment to detect the state of drowsiness. The dataset used is collected from the University of Texas. Various topics related to deep learning has been learned before initiating development. Development is performed using Python in Ubuntu. Results show the performance and limitation of the approach with suggestions for improvement.

This thesis is written in English and is 69 pages long, including 10 chapters, 39 figures and 4 tables.

Annotatsioon

Süvaõppel põhineva sõidukijuhi seisundi jälgimissüsteemi analüüs

Käesoleva lõputöö eesmärk on uurida sõidukijuhi unisuse tuvastamist süvaõppe meetodit kasutades. Kasutati Texase ülikooli andmekogu. Eelnevalt uuriti erinevaid süvaõppega seotud teemasid. Arendus teostati Pythonis Ubuntu keskkonnas. Töö tulemustes on esitatud lahenduse sooritusvõime ja piirangud koos parendusettepanekutega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 69 leheküljel, 10 peatükki, 39 joonist, 4 tabelit.

List of abbreviations and terms

AI	Artificial Intelligence
ANN	Artificial neural network
CNN	Convolutional neural network
CV	Cross Validation
DBScan	Density-based spatial clustering of applications with noise
EEG	Electroencephalogram
ECG	Electrocardiogram
EOG	Electro-dermal activity
HMM	Hidden Markov model
HOG	Histogram of oriented gradients
kNN	K-th nearest neighbour classifier
LSTM	Long short-term memory
MMH	Maximum marginal hyperplane
MTCNN	Multi-task cascaded convolutional networks
MSE	Mean square error
ML	Machine learning
NR	Newton-Raphson method
NHTSA	National Highway Traffic Safety Administration
PCA	Principale component analysis
PERCLOSE	Percentage of eye closure
R-CNN	Region based convolutional neural network
RNN	Recurrent neural network
RSS	Residuals sum of square
RPN	Region proposal network
SVM	Support vector machine
SD	Standard deviation
SPP	Spatial pyramind pooling
VOG	Video-oculo-graphy
VGG	Visual geometry group

3D-CNN	Three dimensional convolutional neural network
UTA-RLDD	University of Texas at Arlington Real-Life Drowsiness Dataset
AUC	Area under the curve
ROC	Receiver operating characteristic
AUROC	Area Under the Receiver Operating Characteristics

Table of Contents

List of Figures

List of Tables

1 Introduction

According to NHTSA reports, in 2017 there were 795 casualties in motor vehicle crashes that involved drowsy drivers. Between 2013 and 2017 there were a total of 4,111 casualties that involved drowsy driving [17] . This work is to design a driver assessment system and evaluate the caliber of deep learning for driver drowsiness detection.

Firstly, the literature review has been performed on a similar solution proposed by the engineering community. Six research paper and one journal is reviewed and presented. The literature review provides information about the various aspect and possible approaches of the present driver drowsiness system. Research has proposed intrusive and non-intrusive approach to collect data about drivers. The collected data is either used directly with mathematical calculation or machine learning models like SVM, CNN, 3D-CNN, LSTM, etc. to detect the state of driver state. Deep learning isn't the new approach for the driver assessment system the literature review provides the aspect of presenting data in multiple dimensions, so the model can see data from multiple perspectives.

Section 4 and 5, include learning curve of the various methodologies, aspect and concept of machine learning. The sections speak about the different types of machine learning, common ML terminologies, various ML model. Section 6 explain artificial intelligence concepts, CNN, RNN, LSTM. These sections incapacitated model selection as per the strength and weaknesses. Made aware of the ways to evaluate model performance. At the end of learning, LSTM model is selected as per its capabilities to use historical predictions to make a new prediction for driver drowsiness detection.

Section 7 explains the data acquisition process. Section 8 explains the LSTM model implementation and evaluation. Python programming language is used for programming. In the end, the result and summary section do the closing with the thesis outcome, dataset shortcoming, improvement suggestion and future work.

2 Literature Review

The paper “*Android OpenCV based effective driver fatigue and distraction monitoring system*"[1] talk about an intelligent system for monitoring driver fatigue and distraction using adaptive template matching and adaptive boosting. First driver drowsiness is identified. Secondly, the eye detection algorithm is proposed to avoid unclear images due to reflection in glasses. Third eye detection accuracy is enhanced by applying eye validation after the initial detection of the eye. Fourth a novel eye state detection algorithm is proposed. The smart phone camera captures video at the resolution of 1280×720 pixels. The video is recorded at the rate of 6fps. Colour Frames resolutions are downsized to 320×180 pixels, and are converted into grey scale images to reduce the processing power and time. Eyes and mouth are detected separately using Haar-like features. Adaptive template matching and adaptive boosting are used to detect the mouth even when the face is rotated. The average duration of eye closure is 400ms, and the minimum duration is 75ms. Therefore if the driver's eyes experience closure more than 400ms then it is considered drowsiness. The Percentage of Eye Closure (PERCLOS) is used to determine whether the driver is feeling fatigued or not.

The paper “*A neural-network-based investigation of eye-related movements for accurate drowsiness estimation*"[2] analyzes two typical eye-related movements, i.e., eyelid movements and eyeball movements, and investigates neural-network-based approaches to model temporal correlations. HMM is used to detect eyes. Eye related motion extracted at 30fps. Hamming filter with window size 11 is used to remove high-frequency noises effectively. Hamming filter is applied separately to eyeball and eye-led, and obtain the filtered signals. Further, to eliminate redundancies, eyeball, and eye-led images are down-sampled to 15 points per second. Use of CNN and RNN over HMM as a result of better performance. The data set is split into five subsets, and the cross-validation is conducted by leave-one-group-out. Video clips of different sizes (10s, 30s, 60s and 120s) are recorded, and the mean of all annotations is taken as ground truth for drowsiness level. Test hardware is Nvidia 1080 8GB with the Adam optimizer.

Batch size to 128 due to fast convergence, and epoch to 100. The learning rate shrinks according to the epoch, starting from 0.001 and decaying by one order of magnitude after 20 and 50 epochs. The evaluation results show that the eyelid movements alone as well as the joint movements; are effective indicators for accurate drowsiness estimation while eyeball movements alone are weakly correlated with drowsiness status. CNN-Net delivers increasingly better results than that of CNN-LSTMNet as video length increases. CNN-Net investigates the short-time correlation from multiple perspectives. On the other hand, CNN-LSTM-Net is designed to remember information along with many time stamps and perform long-term correlation. CNN-Net is more promising to capture short-term eye movements, which are comparatively more vital then long-term eye movements.

The paper "*Driver drowsiness detection using behavioural measures and machine learning techniques: A review of state-of-art techniques*"[3] machine learning techniques for drowsiness detection, which include SVM, convolutional neural networks, and hidden Markov models. There are various measures to determine the level of driver drowsiness. These measures can be grouped into three categories as Physiological Measure, Vehicle-based Measures, Behavioural Measure. Physiological Measures involve EEG, ECG, and EOG. This paper talk about Viola and Jones algorithm, SVM, HMM, and CNN. Here, the face is tracked by a combination of a Kernelized Correlation filter with a Kalman filter for robust face tracking. The extracted face regions are then passed to a 3D-CNN, which is followed by a gradient boosting machine for classification. Performance estimation revealed that CNNs yielded more accurate results when compared to SVMs and HMMs.

The proposed algorithm in the paper "*Drowsy Driver Detection using Representation Learning*"[4] makes use of features learn using the convolutional neural network to explicitly capture various latent facial features and the complex non-linear feature interactions. A softmax layer is used to classify the driver as drowsy or non-drowsy. Convolutional neural nets are a variation of feed-forward neural nets which incorporates three unique features: local receptive fields, sharing of weights and sometimes spatial or temporal pooling. A model consisting of two convolutional layers along with max-pooling operation followed by a hidden layer of sigmoid, which is fully connected to a logistic regression layer for classification. The sigmoid layer applies a non-linear

transformation to the features from convolutional layers. Model is trained using cross-validation by dividing the whole dataset into five folds out of which one fold was used for validation and the remaining four for training. The trained classifier worked efficiently as it gave 92.33% validation accuracy. The average accuracy within subjects was 88% (training and testing for the same person). A satisfactory average result of 78% accuracy across subjects was found in such a case.

In the paper "*Real-time Driver Drowsiness Detection for Embedded System Using Model Compression of Deep Neural Networks*"[5] a heavy baseline model is compared to a lightweight model, deployable to an embedded board. This paper purpose technique to reduce the size of the trained model. To reduce the model size, quantization techniques have been used, such as bit-quantization, binary network, low-rank decomposition, and use of distillation approach between two networks. In distillation approach algorithms, one network plays the role of teacher and another network as a student. Teacher network transfer weights from teacher network to student network using knowledge distillation. In this paper, three types of models are used, which include the baseline 4-stream drowsiness detection model, 2-stream drowsiness detection model, and its compressed version using the teacher-student technique with minimum accuracy drop. The architecture for drowsiness detection involves two steps. The first step is to join face detection and alignment using Multi-Task Cascaded Convolutional Networks (MTCNN), and the second is the drowsiness detection model. Baseline-4 model is a neural network consisting of 5 convolutional layers for each 4-stream input for eyes, face, and mouth. Each stream of the network structure is similar to the AlexNet architecture with filter sizes of 11x 11, 5x 5, 3x 3, 3x 3, and 3x 3. The wide shape of this network makes its size huge and significantly affects its efficiency of speed. Instead of 4 streams in Baseline-4 model, the Baseline-2 model use 2-stream of input, i.e., left eye and mouth. The filter sizes are the same as the baseline-4 model. In terms of performance, there is only 1% reduction in validation accuracy. To further reduce the size of the model without compromising on speed, a model adopts a compression method using the "Distillation" of the neural network. "Distillation" of a neural network refers to an approach transferring the knowledge from a superfluously huge model to a small model. The bigger (teacher) network is trained with huge datasets with the discrete, hard-classified label, which is difficult to do with a smaller (student)

network. The soft value from the teacher network is used to train the student network. In this paper, baseline-2 and compressed-2 models are a teacher and student networks, respectively. For Baseline-4 model face detection and alignment end to end speeds of approximately 6.1fps with Jetson TK1. The model, when run on the test subjects, reported an accuracy of 91.3%. For Baseline-2 model face detection and alignment end to end speeds of approximately 12.5fps with Jetson TK1. The model, when run on the test subjects, reported an accuracy of 92.84%. For Compressed-2 model face detection and alignment end to end speeds of approximately 14.9fps with Jetson TK1. The model, when running on the test subjects, reported an accuracy of 89.5%. In conclusion, baseline-2 model achieves among all the proposed models of 93.8%. Compressed-2 model has the smallest model size.

The paper "*Driver Drowsiness Monitoring System using Visual behaviour and Machine Learning*"[6] use non-intrusive measurement as the sensors are not attached to the driver. In the behavioural method, the visual behaviour of the driver, i.e., eye blinking, eye closing, yawn, head bending, etc. are analysed to detect drowsiness. The face is detected in the frames using histogram of oriented gradients (HOG) and linear support vector machine (SVM) for object detection. After detecting the face, facial landmarks like positions of eye, nose, and mouth are marked on the images. From the facial landmarks, eye aspect ratio, mouth opening ratio, and position of the head are quantified and using these features and machine learning approach; a decision is obtained about the drowsiness of the driver.

System flow stages involve data acquisition, face detection, facial landmark marking, head bending detection, classification. Apart from using thresholding, machine learning algorithms are used to detect drowsiness as well. The EAR, MOR, and NLR values are stored for the synthetic test data along with actual drowsiness annotation. Prior to classification, statistical analysis of the features has been done. First, principal component analysis is used to transform the feature space into an independent one. Secondly, the student's test is used to test whether the features are statistically significant for the two classes. As all the three features are statistically significant at 5% level of significance, all the three features are used for classification using the Bayesian classifier, Fisher's linear discriminant analysis, and Support Vector Machine. The developed algorithm is tested on INVEDRIFAC dataset.

The journal "*Detection and prediction of driver drowsiness using artificial neural network models*"[7] purpose multiple driver drowsiness estimation techniques. These methods are classified in different categories according to the source of information: subjective assessment, sensorimotor indicators, physiological features, and driving behaviour and performance.

Sensorimotor indicators involve reading of eye and head movements. Video-oculography (VOG) is commonly used to study the following features: blink frequency, blink duration, and PERCLOS (PERcentage of eye CLOSure). Physiological features involve the study of electroencephalogram (EEG), the electrocardiogram (EKG), and electrodermal activity (EDA). The relationship between physiological features and cognitive state is hard to define because these physiological features var with other states (including, but not limited to, emotion, workload, physical fatigue) or with the context. Driving behaviour and performance analyses have the main advantage of being non-intrusive. Information about pressure on pedals, car position, or car movements are some common feature used for drowsiness detection. There is no direct link between these features and the "operational state", that is why machine learning is considered.

The goal of this study was to develop and evaluate a model with an artificial neural network (ANN), so as to predict when a given impaired state will be reached in addition to detecting this impaired state. Two hypotheses are considered for the work. First, it is possible to predict when the impaired state will arise by using the sensorimotor, physiological, and performance indicators used to detect drowsiness. Second, we hypothesized that adding information such as driving time and participant information will improve the accuracy of the model.

The ground truth about drowsiness in the video is labelled by two raters. The ANN does processing at one sample per minute for each feature, including ground truth. The modelling has two-part. In the first part, ANN detects the level of drowsiness from the present set of features. In the second part, if drowsiness is less then 1.5, then the second ANN predicts when will drowsiness level reaches 1.5.

The neural network toolbox of Matlab R2013a was used to create the ANNs. Two feed-forward neural networks were used with two hidden layers, and a back-propagation training method was applied using the Levenberg-Marquardt algorithm. To avoid overfitting 'generalization' is used, the total dataset was distributed in a training sub-dataset (70% of the total set, to learn the network's node weights), a validation sub-set (15%: to stop learning and avoid over-training) and a testing sub-set (15%: to evaluate the model's ability to work on previously unseen data. The

ANNs were trained 16 times (4 × 2 × 2) with different datasets combination. Four basic combinations involve three sources of information tested alone or all together; additional two combinations are with or without the elapsed time, and reaming two combinations are with or without information about the participants. The performance function used for learning was the mean squared error, the percentage of numbers of absolute errors below a threshold, the range of errors containing 95% of the values, and coefficient R of the correlation between outputs and targets.

The best performing training combination for drowsiness detection is the one having driving time, the participant information, and behavioural features. With this dataset, the mean square error is 0.22 ± 0.02, and more than 80% of the absolute value of the error of the testing data is under 0.5. Ninety-five percent of the absolute value of the error is under 0.87. Linear regression is applied to represent the correlation graph between real and estimated state. The R-values are close to unity (0.93, 0.91, 0.91, respectively, for the training, validation, and testing datasets). The slopes of the regression lines are very close to unity (0.87, 0.88, 0.88 respectively for the training, validation, and testing datasets), and the intercepts are close to zero (0.17 for all three datasets). As per the frequency histogram for error distribution, the graph show peak at 0.05, implies most of the errors are close to 0. Also, 95% of the instances had and error of between −1.16 and 1.16. Overall, the models perform best for each dataset or for all three datasets used together when both driving time and participant information are included, as shown in Figure 1.

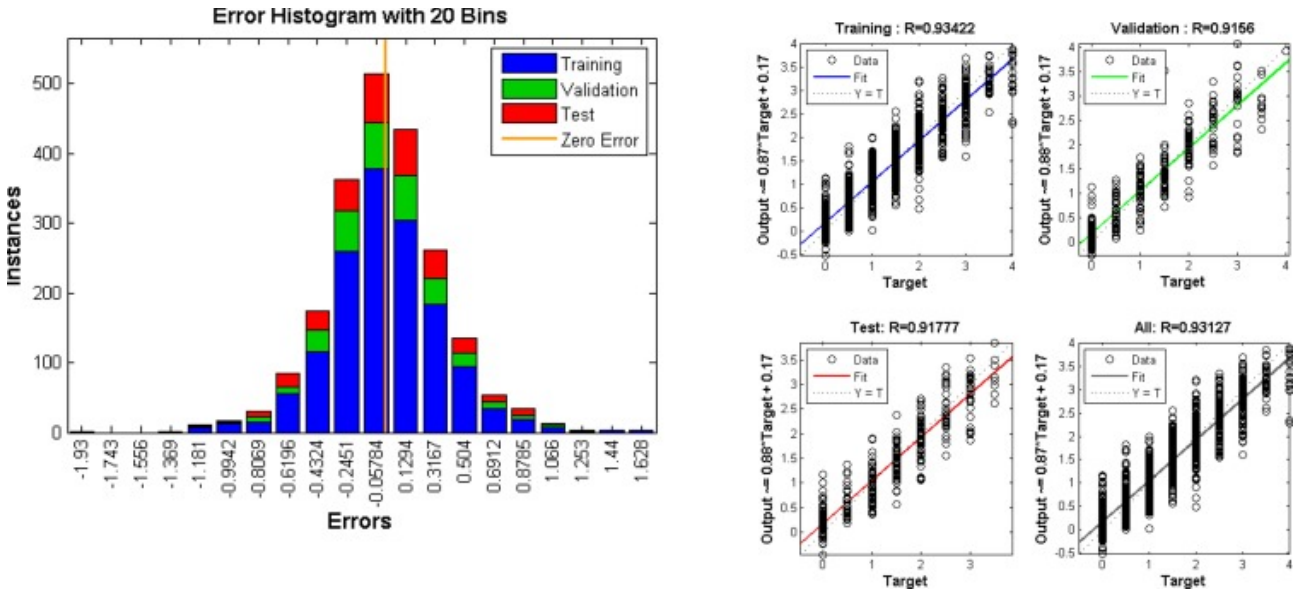

Figure 1. Frequency histogram of error distribution (left panel) and correlation (right panel) between real and estimated state for a model trained with the behavioural dataset, driving time and participant information. [7]

The dataset giving the best performance is the behavioural dataset (use information about eyelid closure, gaze, and head movements) followed by the physiological dataset and finally the car dataset, both in detecting the degree of drowsiness and in predicting when a given drowsiness level will occur. For further improving accuracy, external information such as driving time or a driver profile can be added to the model.

3 Machine Learning

Machine learning gives the ability to computer to learn without being explicitly programmed. Computer ability to predict possible outcome improve with experience.

There are three main types of learning that are supervised learning, unsupervised learning and reinforcement learning [14] .

3.1 Supervised Learning

Supervised learning involves supervising machines in learning, which is called the training phase and then after using this trained model for prediction. So the data set used for training in supervised learning has both influencing variables and outcomes. During supervised learning, the fine algorithm tune itself to match the outcome with the actual results. The algorithm used for supervised learning is regression, naive Bayes, SVM, Neural nets, etc.

An example of supervised learning is to train a model with city medical history and use that information to predict the health-quality or lifespan of the city population for the upcoming year.

3.2 Unsupervised Learning

Unsupervised learning doesn't provide the right outcome to the machine for training. The unsupervised algorithm itself has to find the relationship between influencing variables and form clusters. The algorithm used for supervised learning is k-means clustering, hierarchical clustering, PCA, etc. Genes classification, social marketing are some of the examples of unsupervised learning.

3.3 Reinforcement learning

Reinforcement learning where we do not have a data set at all. The artificial intelligence agent interacts with the environment and figures out what to do to find the solution for the given problem.

There are states, the states represent the environment and there are some actions. The agent makes an action at random to interact with the environment. With the help of this action, the agent goes to another state and that state can be good or bad. The agent receives award for right decision and penalty for the bad decision as shown in *Figure 2*

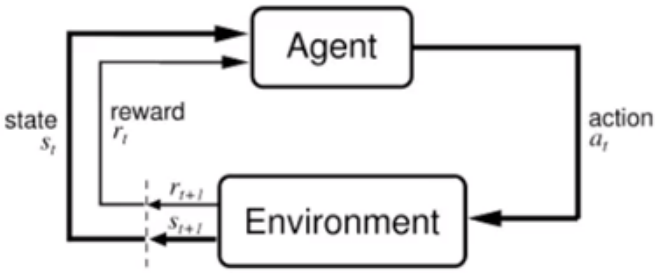

Figure 2. Reinforcement learning state machine.

Every win or loss is equivalent to reward or penalty, respectively. Over the multiple iterations, the agent will perform better in the game.

4 Machine Learning Terminologies

The following section speak about few important ML terminologies before evaluating different ML models that can possibly be used for face detection or drowsiness detection.

4.1 Bias-Variance Trade-off

Bias is the error from misclassification in the learning algorithm. High bias results in under-fitting. Variance is equivalent to error from sensitivity to minor changes in the training set. High variance can cause overfitting. Model complexity increase with an increase in features. So the value of variance and Bias should be computed, keeping model complexity in mind, as shown in Figure 3.

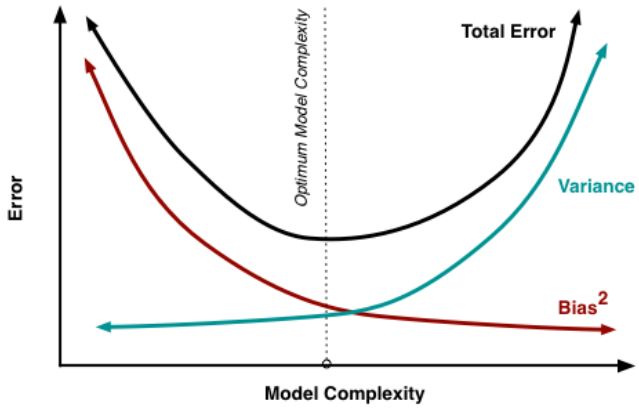

Figure 3. Bias-Variance Trade-off[9] .

4.2 Covariance

The covariance is the joint variability of two random variables (x,y). Covariance for the set of x and y is represented as shown in equation (1).

$$Cov(x,y)=\frac{1}{N-1}\sum_{i=1}^{N-1}[(x_i-\mu_x)*(y_i-\mu_y)] \quad (1)$$

It defines how two variable moves together. A positive value shows that both variables vary in the same direction and negative value shows that they vary in the opposite direc-

tion. The problem with covariance is its dimensional measure and it's not normalized, so it's hard to compare datasets with large standard deviation.

4.3 Correlation

The correlation is a dimensionless measure of how two random variables vary together. Covariance only tells about the direction of change for variables; correlation speaks about the linear relationship. The correlation coefficient varies from -1 to +1. -1 and +1 tell both variables to have a perfectly direct and inverse linear relationship, respectively. 0 means there is no linear relationship between variables; however, there could exist other functional relationships. Correlation between the set of variables x and y is formulated by equation (2).

$$\delta_{xy} = \frac{Cov(x, y)}{\sigma_x \sigma_y} \tag{2}$$

4.4 Confusion matrix

A confusion matrix, also known as the error matrix, is a predictor of model performance on a classification problem. The number of correct and incorrect predictions is summarized with count values and broken down by each class. This lies at the core of the confusion matrix. *Figure 4* represents the confusion matrix; below is the list of terminology present in the confusion matrix.

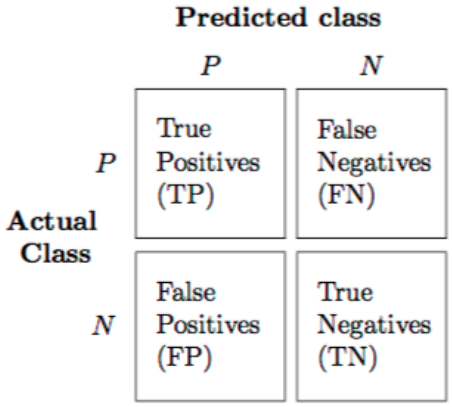

Figure 4. Confusion matrix.

True Positive (TP): This refers to the cases in which we predicted “YES” and our prediction was actually TRUE.

True Negative (TN): This refers to the cases in which we predicted “NO” and our prediction was actually TRUE.

False Positive (FP): This refers to the cases in which we predicted “YES”, but our prediction turned out FALSE.

False Negative (FN): This refers to the cases in which we predicted “NO” but our prediction turned out FALSE.

4.5 Overfitting

The overfitting model has trained "too well" on the training dataset. It means it's very accurate on the training dataset but yields poor results on the test set. This happens when the model is too complex, and the model learns noise instead of actual relationships between the variables present in training data. Figure 5 show graphical representation of overfitting.

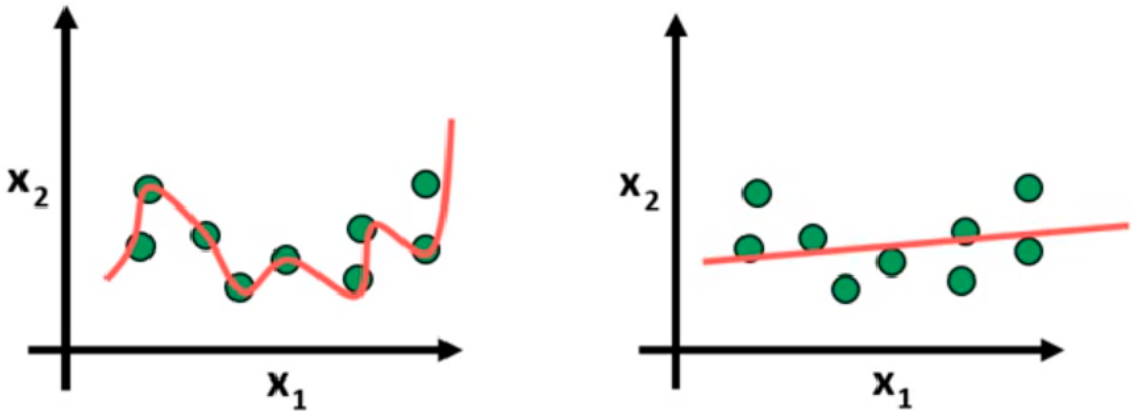

Figure 5. Overfitting and Underfitting.

4.6 Underfitting

The Underfitting model has not been fitted well to the training dataset, results in missing the trends in the training dataset. Usually, this is the case when the models for a problem is too simple. Figure 5 shows the graphical representation of underfitting.

4.7 Cross Validation

The process of deciding whether the numerical results quantifying hypothesized relationships between variables on bases of accuracy and variance are acceptable as descriptions of the data is know a validation. Cross validation is a re-sampling procedure used to evaluate a model when we have limited data. There are two approaches in CV. Firstly, Train-Test split approach, in which datasets are divided into 70:30 or 80:20 ratio for training and testing, respectively. Secondly, K-Folds CV, the datasets are split into k folds. In every train-test cycle, k-1 folds will be used for training and 1 fold for testing. This way, every fold will be part of the training and testing process. CV helps to avoid underfitting and overfitting problems.

4.8 Grid Search

Grid search is a methodology to perform hyper-parameter optimization, i.e., it's a method to find the best combination of hyperparameters for a given model and test set. Grid search applies a different combination of hyper-parameters to prepare, unlike models. Each of these, unlike models, corresponding to a point on a "grid". Each of these models is trained and evaluated in order to find hyper-parameters giving the best performance.

4.9 Principle component analysis (PCA)

The purpose of the principal component analysis is to reduce the dimensionality of the dataset by finding the possible similarities between features, at the cost of minimal information loss. This way computational cost and parameter estimation error can be reduced.

In order to find PCA, the first correlation between the dataset has to be calculated by finding the covariance matrix of the data set; this is the groundwork for PCA. Figure 6 show how a correlated dataset looks like for a Gaussian distributed data. On the left side of the image in the matrix, the diagonal elements show the variance, and off-diagonal elements show the covariance. White box show maximum variance, black shows no relationship between data being compared and red show moderate relationship. The covariance matrix provides a best-fitted line for a correlated dataset. Figure 7 shows the best fit line in red, which is equivalent to the covariance matrix for a dataset.

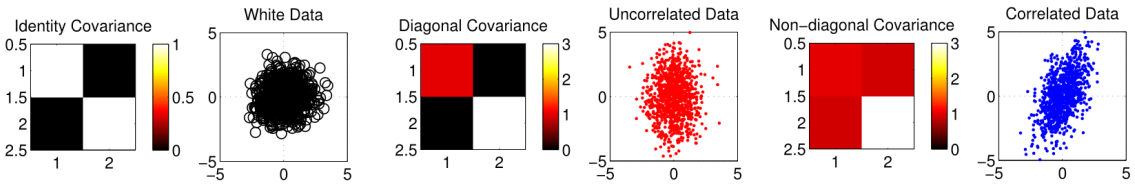

Figure 6. Covariance matrix for different kind of dataset

The properties of the covariance matrix can be further investigated using Eigenvector and Eigenvalue. The Eigenvector will be the unit vector in the direction of the best fit line and Eigenvalue is the sum of the square of the distance between origin and shadow of the data point. Ideally, the number of Eigenvector will be equivalent to the number of dimensions and describe about PCs, but in order to reduce the computation, the Eigenvector count can be limited to some of the those having highest Eigenvalue. This signifies the level of information Eigenvector carries about the distribution of data. All the PCs are perpendicular to each other and have the shadow of actual data points. Selected PCs along with there Eigen properties are used as the new dimensional axis. The shadow point from PCs are used to deduce new features data points.

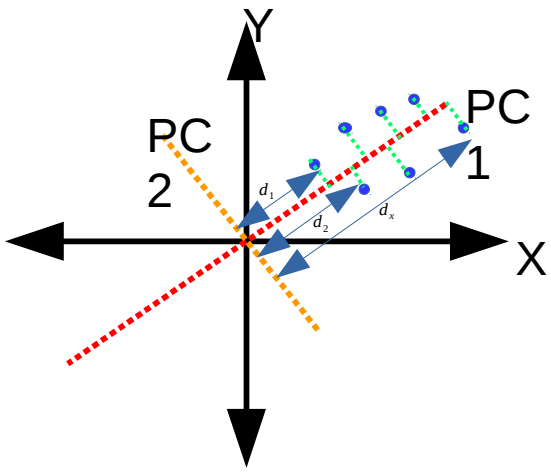

Figure 7. Graphical representation of principle components

5 Classification Models

Multiple classification models are been studied before switching to deep learning approach. This section talks about classification models. This study is performed to understand the significance of the classification models and requirement of deep learning approach.

5.1 Logistic Regression

Logistic regression is used for the classification problem when the outcome of the dependent variable is discrete. It assigns probabilities to given outcomes, which signify whether or not output belongs to a certain class.

Linear regression can't be used for the classification problem because it's output would possible outside the classification range. Linear regression is sensitive to outliers and it doesn't deal with probabilities. Logistic regression overcomes this problem using the sigmoid function. Equation (3) represent the sigmoid function.

$$f(x)=\frac{1}{1+e^{-x}} \tag{3}$$

So, the probability of default given x is given by equation (4).

$$p(x)=P(y=1|x;b)=\frac{1}{1+e^{-(b_0+b_1*x)}} \tag{4}$$

The find optimized b parameters, either we can use gradient descent or maximum-likelihood method. The sigmoid function is not linear, but the logit transformation of the estimated probability response is linear, as shown in equation (5), this behaviour is called generalized linear model.

$$\ln\left(\frac{p(x)}{1-p(x)}\right)=b_0+b_1x \tag{5}$$

Equation (4) consists of the non-linear sigmoid function, which makes it a non-convex function, therefore it's possible that gradient descent or maximum-likelihood method find local extrema instead of global extrema. We need a convex cost function [8] . The

probability mass function for y = 0 logistic regression can be represented using equation (6), (7) and (8).

$$P(y=1|x;b)=H_b(x) \tag{6}$$

$$P(y=0|x;b)=1-H_b(x) \tag{7}$$

$$\text{here} \quad H_b(x)=\frac{1}{1+e^{-z}}, z=b_{0x}+b_1 \tag{8}$$

Equation (6) and (7) can be combined to result in a simplified cost function.

$$P(y|x;b)=H_b(x)^y(1-H_b(x))^{(1-y)} \tag{9}$$

For a vector type input, the likelihood function is applied to every sample to get cumulative likelihood function, represented in equation (10) and (11).

$$L(b)=\prod_{i=1}^{n} p(y_i|x_i;b) \tag{10}$$

$$L(b)=\prod_{i=1}^{n} H_b(x_i)_i^y(1-H_b(x_i))^{(1-y_i)} \tag{11}$$

Multiplying “n” likelihoods together, produce a result of magnitude 10^{-n} , this can lead to loss of precision during the calculation. The solution to this problem is the log of likelihood function called log-likelihood of hypothesis function, as shown in equation (12) and (13).

$$l(b)=\log L(b) \tag{12}$$

$$l(b)=\sum_{i=1}^{n} y_i \log(H_b(x_i))+(1-y_i)\log(1-H_b(x_i)) \tag{13}$$

Equation (13) is a convex function and by using gradient descendant or the newton-raphson (NR) optimization, exterma can be found. Newton-raphson finds exterma of a

function in an iterative manner. Equation (14) show the recipe, starting from a point x_n to the next point $x_{(n+1)}$ in the iterative series until $x_n - x_{(n+1)} \approx 0$.

$$x_{(n+1)} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{14}$$

For a multi-dimensional input, single-variable derivatives are replaced with a partial derivative called the gradient. For the second-order derivative, take the partial derivative of each first-order partial, with respect to each parameter. The Hessian is a square matrix of second-order partial derivatives of order nxn. The overall newton optimizer iteration for multi-dimensional input looks like an equation (15).

$$b_{(n+1)} = b_n + H^{-1}_{(l(b))} \nabla l(b) \tag{15}$$

here $\nabla l(b)$ is the gradient of log-likelihood and H is Hessian matrix for second-order partial derivative of log-likelihood.

5.2 K-th Nearest Neighbour Classifier

K-nearest neighbours classifiers can classify objects by assigning them the class of the most similar object examples. kNN is suitable for classification problems where the relationship between the features is very complex. The unlabelled test object is assigned to the class of the majority of the k nearest neighbours.

This algorithm is called lazy learners because it doesn't perform any training. This impersonates that training is very fast, but performing prediction is rather slow since, during prediction algorithm, calculate euclidean distances. Equation (16) represent euclidean-distance.

$$Distance(x, y) = \sqrt{((x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2)} \quad (16)$$

The most important part of kNN is to determine how well the model will generalize and perform on the new dataset. If k is small, it causes over-fitting because it models the noise and becomes a high variance and less biased model. If k is high, it causes under-fitting because variance is small, but on the other hand, there is high bias.

Consider Figure 8 shows the distribution of a random dataset. It has a blue and red class. To predict green data point belongs to which class, euclidean-distance has to be calculated between the new data point and points belong to the training dataset. If K=3, then a new point belongs to the red class because two out of three data-item are labelled to red class. If K=5, then the new point belongs to blue class because three out of five data-item are labelled to blue class.

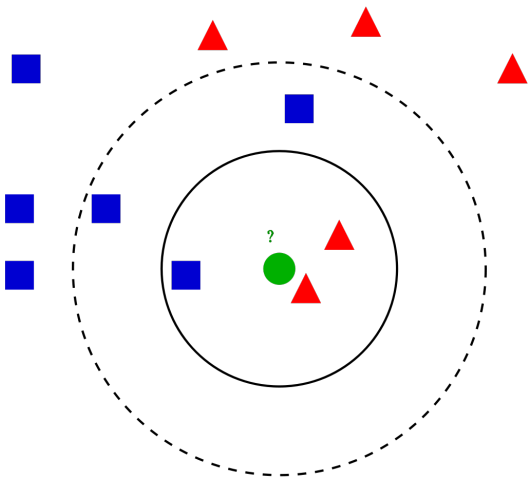

Figure 8. Data set distribution for kNN.

5.3 Support Vector Machine

Support vector machine is a classification algorithm. The basic idea behind SVM is to find a decision boundary (line/hyperplane) with the widest margin as possible to separate data belonging to a different class, as shown in Figure 9. One magical feature of SVM is “Kernal Trick”, if a dataset non-separable, then transforms data to a higher dimension in order to find a decision boundary. Before seeing SVM in detail, the following components have to be explained.

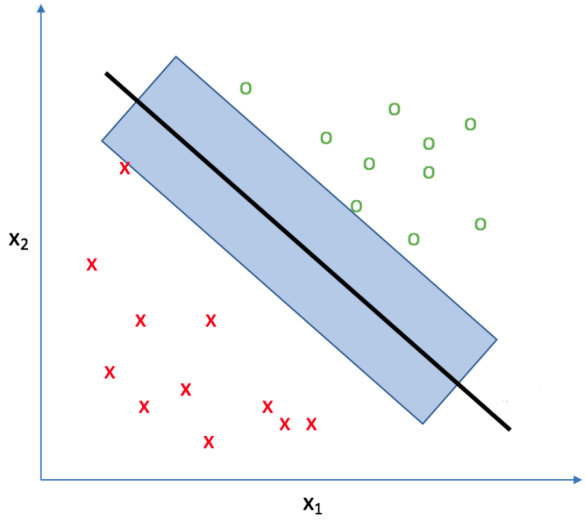

Figure 9. Support Vector Machine classification for 2 dimensional data set.

5.3.1 SVM optimization derivation

Consider we have data set as shown in Figure 10, with separation-line/hyperplane shown by the doted-black line and boundary shown by the orange line. The orange line is drawn to have the widest separation margin between positive and negative samples, and this is called the “Widest Street Approach”. Now the motive is to find a decision rule which uses the hyperplane.

Figure 10 show vector $\bar{w}$ starts from origin and perpendicular to the boundary. There is a random vector $\bar{u}$. To find whether $\bar{u}$ belong to the positive or negative class, project $\bar{u}$ on $\bar{w}$. Mathematically this is called dot product between $\bar{u}$ and $\bar{w}$ represented by equation (17).

$$\bar{w} \circ \bar{u} + b \geq 0 \tag{17}$$

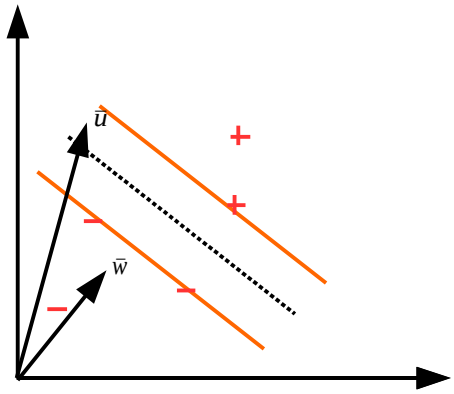

Figure 10. Graphical representation of decision boundary and margin.

If the condition in equation (17) fulfilled, then the sample $\bar{u}$ belongs to the positive class. “b” is the constant in the equation to maintain the generality. Equation (39) is the decision rule but $\bar{w}$ and “b” has to be found. To derive $\bar{w}$ and “b”, there is a need to create more constraines. Equation (18) and (19) are two constraines for sample belongs to positive and negative class.

$$\bar{w} \circ \bar{x}_{+} + b \geq 1 \tag{18}$$

$$\bar{w} \circ \bar{x}_{-} + b \leq -1 \tag{19}$$

For mathematical convince, a new variable y_i is introduce with following properties.

$$y_i = \begin{cases} +1 \text{ for } \textit{postive sample} \\ -1 \text{ for } \textit{negative sample} \end{cases}$$

Multiplying y_i to equation (18) and (19) gives equation (20), which is also know as “Funtional margine” equation.

$$y_i(\bar{w} \circ \bar{x}_i + b) - 1 \geq 0 \tag{20}$$

Equation (20) carries the properties of y_i for samples outside the decision margin. For samples on decision margin, the decision equation will look like equation (43).

$$y_i(\bar{w} \circ \bar{x}_i + b) - 1 = 0 \tag{21}$$

The purpose of optimization is to have maximum distance between margin. In Figure 11, there are two arbitrary sample $\bar{x}_{+}$ and $\bar{x}_{-}$ show that the distance between margin

is calculated by dot product of $\bar{x}_+ - \bar{x}_-$ and unit normal vector of $\bar{w}$. Same is relation is depicted in equation (22), using property of y_i , $\bar{w}$ and equation (21).

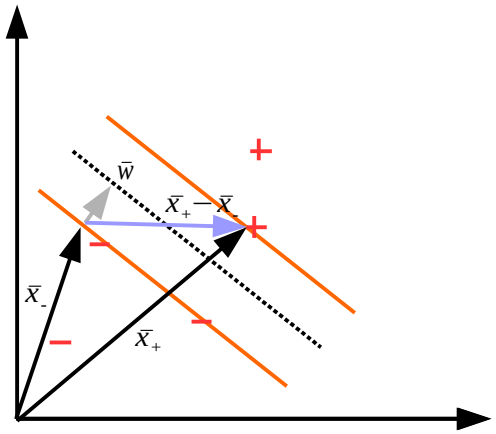

Figure 11. Distance between margin.

$$max\left((\bar{x}_+ - \bar{x}_-)\circ\frac{\bar{w}}{\|w\|}\right) = max\left(((1-b)-(1+b))\circ\frac{\bar{w}}{\|w\|}\right) = max\left(\frac{2}{\|w\|}\right) \approx max\,\frac{1}{\|w\|}$$
$$max\,\frac{1}{\|w\|} = min\,\|w\| \approx min\left(\frac{1}{2}\|w\|^2\right) \tag{22}$$

In nutshell, the learning task in SVM, can be formulated as following constrained optimization problem to minimize $\mu(w,b) = \left(\frac{1}{2}\|w\|^2\right)$ subject to $y_i(\bar{w}\circ\bar{x}_i + b) \geq 1, i = 1,2,3...n$. To find the extremum in order to maximize the distance between the margin keeping the constrain in mind, Lagrange multiplier comes under picture.

$$max\,L = max\left(\frac{1}{2}\|w\|^2 - \sum \alpha_i [\, y_i(\bar{w}\circ\bar{x}_i + b) - 1)\,\right] \tag{23}$$

To find the extremum, derivation has to be performed. For vector in equation (23) it will be partial derivative.

$$\frac{\partial L}{\partial \bar{w}} = \bar{w} - \sum \alpha_i y_i \bar{x}_i = 0 \quad \text{here} \quad \bar{w} = \sum \alpha_i y_i \bar{x}_i \tag{24}$$

$$\frac{\partial L}{\partial \bar{b}} = -\sum \alpha_i y_i = 0 \tag{25}$$

This turn out to be quadratic optimization problem. Using equation (23), (24) and (25) the equation (26) and then further equation (27) is derived.

$$L=max(\frac{1}{2}(\sum \alpha_i y_i \bar{x}_i)\circ(\sum \alpha_j y_j \bar{x}_j)-(\sum \alpha_i y_i x_i)\circ(\sum \alpha_i y_i x_i)-\sum \alpha_i y_i b+ \tag{26}$$

$$L=max(\sum \alpha_i-\frac{1}{2}\sum \alpha_i \alpha_j y_i y_j x_i \circ x_j) \tag{27}$$

here i and j represent a pair of observations present in the dataset. From equation (27) we can say that the optimization of cost function depends on the dot product of support vector and test sample. From equation (17) and (24), the decision rule will look like equation (48), and it represents the significance of dot product for optimization equation.

$$\sum \alpha_i y_i \bar{x}_i \circ \bar{u}+b \geq 0 \tag{28}$$

Value fulfilling equation (28) belong to positive class of Figure 11. Once $\bar{w}$ is calculated, then respective value can be use to find "b". From equation (27) following can conclude:

i) $\alpha_i \alpha_j$ determine whether or not the corresponding pair play a significant role in deciding the decision boundary.

ii) $y_i y_j$ compare the output label for x_i and x_j . If x_i and x_j belong to the same class, then the product will be one otherwise zero.

iii) $x_i x_j$ is representation of how similar the vector x_i and x_j are to each other.

In case data is not linearly separable, then instead of directly using the feature, those can be transformed into a higher dimension using Kernel functions. Transforming all the samples to higher dimensional is computationally expensive. Kernel functions operate in high-dimension implicitly without ever computing the coordinates of data in that space. Some of the common kernel functions are the Linear kernel, Polynomial kernel, Radial basis function, etc. One more problem with linearly inseparable data set is the possibility of over-fitting in case of transformation to a higher dimension. So, a possible solution is a soft margin. In real-life problems, it's common to have outliers, as shown in Figure 12. To consider outliers, slack variable ξ_i included in the constraints of the optimization problem, as shown in equation (29). Higher the value of ξ_i more strictly the constrain will be followed. This is also called regularization. Equation (29) show the optimization problem with regularization parameter "C" and Figure 10 depict very-high

and very-low value of “C” can result into incorrect MMH. “C” determine how vital the value of ξ_i is.

$$\min_{(w,b,\xi)} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \quad \text{subject to}$$
$$y_i(\bar{w} \circ \bar{x}_i + b) \geq 1 - \xi_i, xi_{¿} i \geq 0, i = 1,2,3...n \tag{29}$$

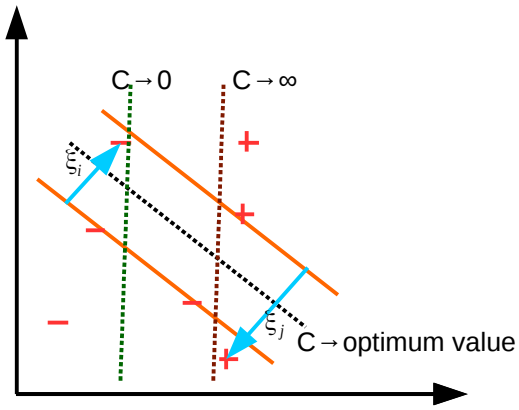

Figure 12. Soft margin scenario for SVM.

In conclusion, the basic idea of SVM is to find MMH. In order to find MMH, it has been considered that there exists a hyperplane with a maximum margin. It's known that hyperplane/decision-line depends on $\bar{w}$ and b. It's also known that hyperplane has constrained its classification result as per the position of the sample. So, the values of $\bar{w}$ and b should be such that it fulfill the constrain. For MMH, width of margin should be maximum. The margin width will be maximum only if we achieve the minima of the cost function. SVM cost function also includes constraints. Langrang multiplier help to tune $\bar{w}$, b and α to find minima of such a cost function. As a result, equation for $\bar{w}$, b and α is achieved, which is tuned using training data set to find a relation for classification of class.

6 Artificial Intelligence

Whole idea behind this work is to evaluate AI for driver status monitoring system. This section talks about the AI concept and models which can be possible used for developing driver status monitoring system.

For machines, it's easy to solve optimization problems using numerical methods. But the problem which can't be reliably represented by the mathematical formula, for example, facial recognition, natural language processing, etc. but not human emotions. For humans, these can be easy problems but not for the machines. Artificial Intelligence mimics the human brain in order to learn and try to solve problems like humans. Inspired by the human nervous system, each neuron is represented by a node and axon by branch or edge between nodes, results in a unidirectional or bidirectional graph, as shown in Figure 13. The neural network learns by changing the weight across the edge. In combination with the traditional machine learning technique like SVM, linear classifiers, etc., AI outperforms orthodox approaches.

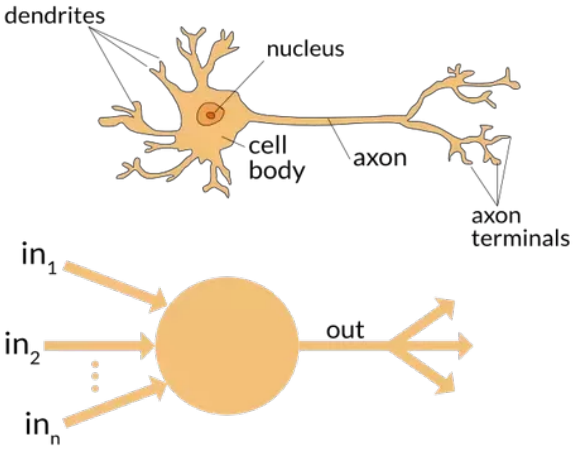

Figure 13. Single neuron in neural net compare to a neuron in human nervous system.

6.1 Neural Net

Figure 14 shows the typical example of a neural net which involve a series of neuron along with the bias unit. There is an input layer equal to the number of numerical features, and initial data processing is performed on this input layer. Second is the hidden layer consist of perceptrons needed for making a prediction for non-linear problems. Lastly, the output layer for the number of possible results. The information is controlled by edge weight. So over the multiple iterations of training, edge weight will be modified using back-propagation to get quality model [11] . In this section important terminologies of AI is discussed.

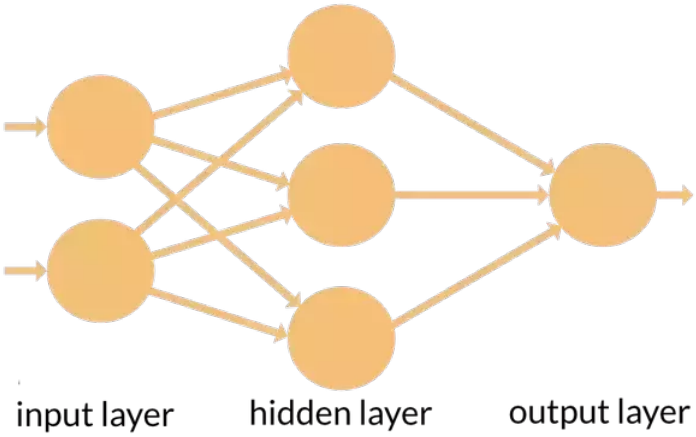

Figure 14. Representation of neural net.

6.1.1 Perceptrons

The perceptrons involve summing the product of input and edge weights belong to individual neurons and then using activation function for the prediction. The result of the activation function is either 1 or 0. For sigmoid-neuron, the results vary between 0 and 1. The sigmoid-neuron and perceptrons level of sensitivity to stimulus is depicted by the threshold values of activation function and accordingly, it can be decided either to use sigmoid-neuron or perceptrons as per the scenario.

6.1.2 Bias node

Bias nodes are to shift the output of the activation function. For example, for an ideal sigmoid activation function, the output is as shown in Figure 15. The steepness of the sigmoid varies as per the weights, Figure 16 shows the same for different edge weights. Using the bias node, the output can be manipulated to desire representation by adding or subtracting the product of the bias node and it's respective weight, as shown in Figure 15.

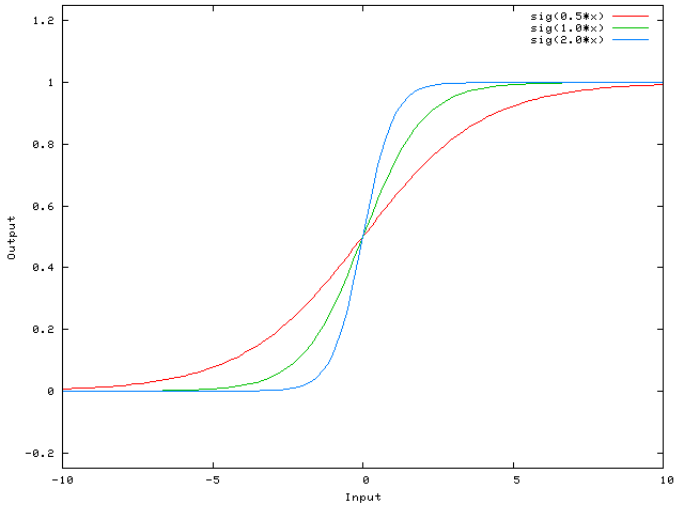

Figure 15. Sigmoid with different weights.

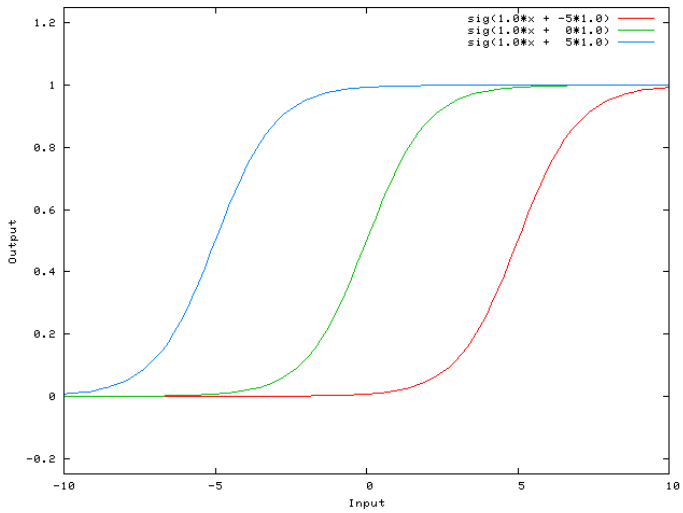

Figure 16. Sigmoid with different weights and bias.

6.1.3 Activation Functions

Activation functions are responsible for making the decision as per there characteristic for the given input. The input layer neuron of the deep neural network doesn't perform any data manipulation, so these are said to have linear activation functions. The hidden and the output layer neuron are equipped with non-linear activation functions because the real-life dataset is mostly non-linear. In following section, various activation functions are analysed in order to select one for drowsiness classification.

6.1.3.1 Sigmoid

Equation (30) represents the sigmoid function. These are mainly used only in the feed-forward neural network because of there limitation of small derivative in back-propagation.

$$\hat{y}=\frac{1}{\left(1+e^{(-x)}\right)} \tag{30}$$

6.1.3.2 TanH

Equation (31) represents the TanH function. The curve is similar to Sigmoid, but it's a centre lie at the origin. This function is favourable to the dataset having a significant amount of negative as well as positive samples.

$$\hat{y}=\tanh\left(x\right)=\frac{\left(e^{x}-e^{(-x)}\right)}{\left(e^{x}+e^{(-x)}\right)} \tag{31}$$

6.1.3.3 Relu

Equation (32) represent the Relu function. The Relu stands for Rectified Linear Unit. The rise of Relu is that it's derivative is either zero or one, which overcomes the limitation of Sigmoid and TanH for back-propagation. Figure 17 show the Leaky-Relu funtion characteristic.

$$\hat{y} = \begin{cases} 0 \text{ for } x \leq 0 \\ x \text{ for } x > 0 \end{cases} \tag{32}$$

6.1.3.4 Leaky-Relu

One drawback with Relu is edge-weight get nullify during back-propagation if the input is less then zero because of it's zero derivative value. To overcome this characteristic of Relu, Leaky-Relu multiplies a constant for input value less than zero, which gives a non-zero slope and save edge-weight from becoming zero. Figure 17 show the Leaky-Relu funtion characteristic. Equation (33) represents the Leaky-Rely function, here C is the non-zero constant.

$$f(y) = \begin{cases} C * y \text{ for } y \leq 0, C \neq 0 \\ y \text{ for } y > 0 \end{cases} \tag{33}$$

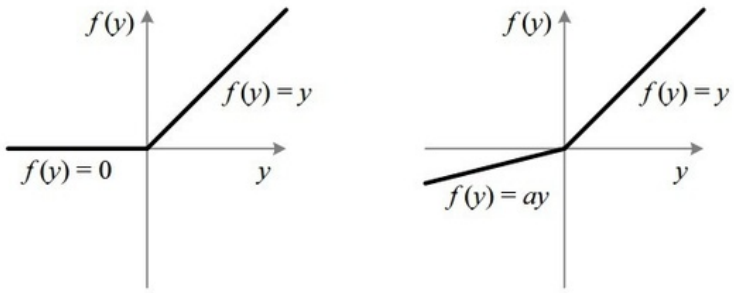

Figure 17. Relu and Leaky Rely characteristic.

6.1.3.5 Softmax

Equation (34) represents the Softmax function. The output layer softmax activation function is suitable for multiple class classification problems. Because of its normalization property, it assigns probabilities distribution to each of the possible output classes.

$$S(f_{yi}) = \frac{e^{(f_{yi})}}{\sum_{j=0}^{i} \left(e^{(f_{yj})}\right)} \tag{34}$$

Softmax, in tandem with negative log-likelihood loss function, provides the accuracy of the predicted sample. Figure 18 shows the graphical characteristic of the negative log-likelihood. If the correct class has higher probabilities, then the loss value will be minimum.

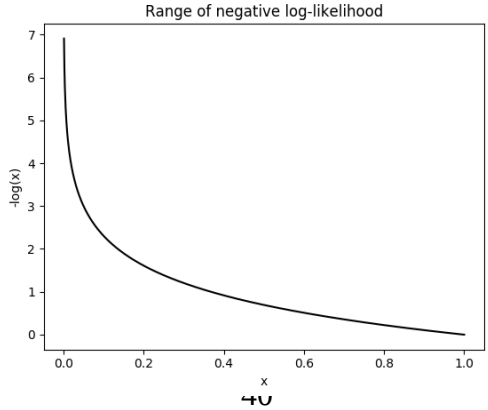

Figure 18. Negative log likelihood (-log(i)) character.

6.1.4 Optimization

Figure 28, shows a feed-forward neural network with respective weight for each edge node. Edge weight has to be optimized to obtain precise output, mathematically it's represented by the cost function shown in equation (35).

$$C(w)=\frac{1}{(2n)}\sum_{i=0}^{n}\|f(x)-y\| \tag{35}$$

y is the output of the neural network, f(x) is the known output from training dataset resulted from the input and assigned weights, n is the number of input. Higher the value of cost function, more the rework is required to reduce the error and to find the minima of the cost function. Gradient descends in combination with back-propagation is used for finding global extrema of cost function and optimizing edge weight.

For gradient descend, MSE is considered as the cost function, as shown in equation (35). In finding minima, a cost function is converging using a partial derivative. Convergence behaviour toward minima influenced by derivative property of activation function, actual input, expected output and edge weight. The relation between error and activation function is represented by . Equation (36) represent the overall converge flow of MSE cost function.

$$\begin{gathered}\frac{(\partial C)}{(\partial w)}=\frac{(\partial MSE)}{(\partial w)}=\delta*input=(MSE*(\frac{(d(sigmoid))}{(d(input))}))*input \\ (MSE*(\frac{(d(sigmoid))}{(d(input))}))*input=(C(w)*(\frac{(d(sigmoid))}{(d(input))}))*input\end{gathered} \tag{36}$$

The process of using convergence flow in order to update edge weight to get the minimum error in the result is called back-propagation. Equation (37) represent the back-propagation.

$$\Delta w_i=\alpha*(\frac{(\partial MSE)}{(\partial w_i)})+(\mu*\Delta w_{(i-1)}) \tag{37}$$

Δw_i and $\Delta w_{(i-1)}$ represent the change in weight for present and previous iteration, respectively. The size of convergence steps is controlled by the learning rate i.e. α . The speed of convergence is controlled by the momentum μ . Higher μ can result to overshooting even having small a value of α and lower μ can cause local minima. So it takes regression to find appropriate value of μ and α .

RResilient propagation is also an option to find the optimum value of edge-weights using gradient descendants. But compare to back-propagation in resilient propagation, only focus on the sign of the weight variation call Δw_i , signify the direction of the gradient is converging. If convergence is in the same direction, then the learning rate and weight can be modified in real-time to speed up the convergence.

6.1.5 Neural network learning

Figure 19 show the learning flow for neural network.

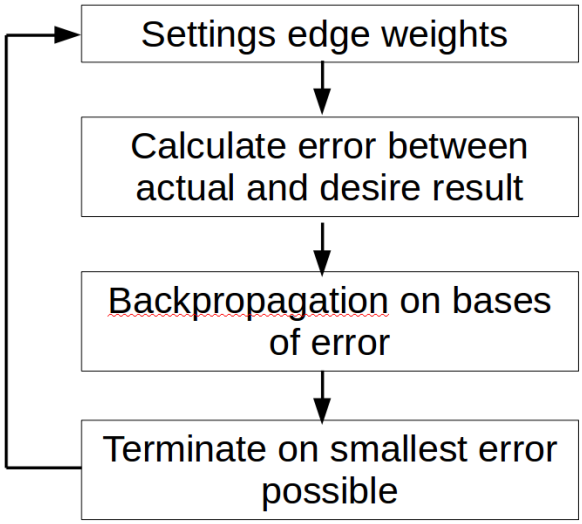

Figure 19. Neural network learning flow.

6.2 Deep Neural Net

The neural net with multiple hidden layers is called a deep neural net. There isn't any universally acceptable reason for the better performance of deep neural net compared to the shallow one. Deep neural net help to modularize and sentimentalize the learning process results in a smoother result. The wall of the shallow net will need a lot of external stimulation in order to match the deep neural net. The count of the neuron isn't the only factor of performance; otherwise, elephants would be smarter than humans. But anyhow, it's statically proven that deep neural net performs better than the shallow neural net for various types of problems. Apart from the first and last layer of deep net, the training isn't similar to the shallow net. Using sigmoid and tanh activation function cause vanishing gradient problem within the hidden layer. As deep neural net is used for

developing driver drowsiness detection, following section explain aspects of deep neural net to be taken care.

6.2.1 Vanishing Gradient Problem

Equation (37) show the dependency of activation function derivation for the optimization of cost function. In a deep neural net the order of the activation function derivation is directly proportional to the number of the hidden layer. The higher level derivative of traditional non-linear activation functions like sigmoid and tanh degrades, as shown in Figure 20, result decrement in the change in weights. With each new hidden layer, the quantity of weight change decreases. This leads to a point where updating in edge weight is approximately zero, and the purpose of gradient descendant to find global minima vanish away. This reduces the overall efficiency of the model training using back-propagation.

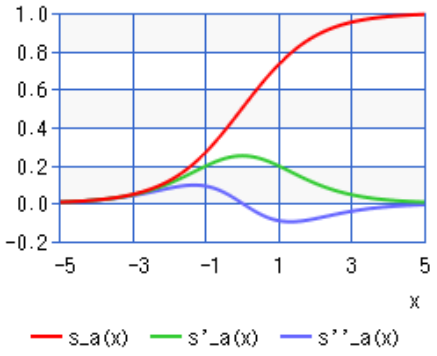

Figure 20. Sigmoid funtion 1st, and 2nd level derivation.

The solution to the vanishing gradient is to replace the hidden layer sigmoid or tanh activation function with the family of relu functions. The derivative of Relu function either results in zero or one. If it's one, then the value for sure change in weight will surely be considered. If it's zero, then there won't be any change in weights. To overcome the scenario of zero derivative, Leaky-Relu activation functions can be used. Figure 21 shows the first degree derivative of Leaky-Relu; for input below zero, the derivative result can be tuned with some constant to avoid zero.

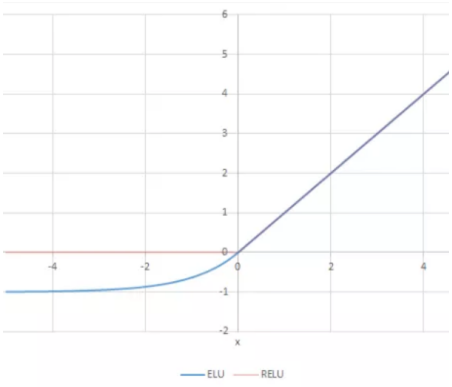

Figure 21. The first-order derivative of Relu and Leaky-Relu.

6.2.2 Exploding Gradient Problem

Appropriate weight parameter selection in back-propagation is the vital otherwise higher value of weight result to exploding gradient problem. In deep neural net, weight updation involve a chain of the partial derivation of each layer activation function. Each layer of weight updation depends on the initial value of weight. If the initial value is too high, then the absolute value for change is weight result to very large cause large irrelevant jumps of cost function without achieving global minima. There are different tricks to initialize weights as per the activation function, for example, normally or uniformly distributed random value with mean at zero and variance of one, “Xavier” for Sigmoid, “He init” for Relu, etc.

6.2.3 Over-fitting

One of the common problems with the deep neural net is overfitting because of the big-data and lot of tunning parameters. So either train the model until it overcomes overfitting or changing the complexity by changing the structure or network parameters. The following techniques can be used for deep learning model to overcome over-fitting.

6.2.3.1 Data Augmentation

Representing dataset from different prospectives or adding noise to dataset keeping labels to data. Training this way makes model confident to it’s result within a limited amount of dataset. For example, in image processing, rotating the image or distorting the image will serve the purpose.

6.2.3.2 Early stopping

Figure 22 shows the typical scenario where early stopping will be suitable. Early stopping performs cost function optimization while being concern about over-fitting simultaneously. Early stopping halts the training where the model is performing the best. This gives the suitable weights which reduce the overfitting. The conditions which define the desire performance should be selected as per the dynamics of the problem. For example, it can based on the result comparison of last and present epoch or using average values or having some offset before making the decision on result comparison to avoid fluctuation, etc.

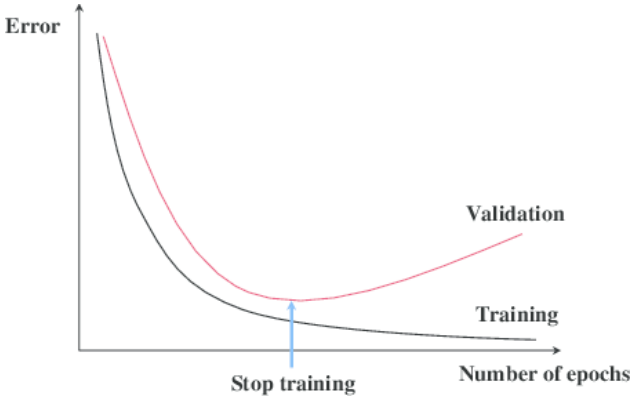

Figure 22. Training and validating property of a model for early stopping.

6.2.3.3 Weight regularization using L1 and L2

L1 and L2 weight regularization use the norm of the weight vector set to penalize the weight as per it's significance in order to avoid overfitting while training. Equation (38) and (39) show the norm for L1 and Ln, respectively.

$$\|w\|_1 = |w_1| + |w_2| + ... + |w_n| \tag{38}$$

$$\|w\|_n = \left(w_1^2 + w_2^2 + ... + w_n^2\right)^{(1/n)} \tag{39}$$

Considering equation (35) as the cost funtion, equation (40) show loss function with L2.

$$L(w) = (\hat{y} - C(w))^2 + \lambda \sum_{i=1}^{n} w^2 = H(x) + \lambda \sum_{i=1}^{n} w^2 \tag{40}$$

here $\hat{y}$ is the actual output, λ is the regularization coefficient varies greater then 0 and w is weight associated with the nodes in the neural net. For the initialization select λ as equal to the inverse of the number of samples[7]. The optimization of λ will be the part of hyper-parameter optimization in training procedure.

Gradient descendant is used to optimizing the weight in order to minimize the error, so L2 norm contribution to equation (37) results in equation (41).

$$\Delta w_i = \alpha * \left(\frac{(\partial\, MSE)}{(\partial\, w_i)}\right) + (\mu * \Delta w_{(i-1)}) + 2\lambda\, w_{(i-1)} \tag{41}$$

here $w_{(i-1)}$ make sure the polarity of λ is as per the momentum.

It's is know that cost function without regularization will lead to over-fitting; that means with the addition of L2 regularization there is room for improvement by relaxing the perfect model from hard-coded inputs.

6.2.3.4 Dropout

The idea behind drop-out is at every combined iteration of feed-forward and back-propagation, randomly activate the number of nodes at each layer as per the drop-out ratio, which decides active and de-active nodes. Vote out results from overall output to find the right result.

6.3 Batch Normalization

In deep learning, batch normalization is used to restrict the input value range for the input layer by limiting the variance and mean for input and deep layers hyper-parameters. The mean and variance for the input layer will 0 and 1, respectively.

Equation (42) and (43) show batch normalization for the inputs of a neuron.

$$z^{i}_{norm} = \frac{(z^{i} - \mu)}{\sqrt{(\sigma^{2} + \varepsilon)}} \tag{42}$$

$$\tilde{z}^{i} = \gamma\, z^{i}_{norm} + \beta \tag{43}$$

z^{i} is the summation of the product of weight input to i^{th} neuron of a layer, μ is the mean of the input to the neuron, σ is the variance of the input to the neuron, ε is the constant to avoid zero variance. Equation (42) have mean 0 and variance 1. Equation (43) have tunable hyperparameter γ and β parameter to control mean and variance. For the hidden layer, it's not suggestible to maintain mean 0 and variance 1 while optimizing weight because it can limit the convergence performance of the gradient descendant.

Batch normalization also performs regularization unintentionally by the noise present in the mean and variance of dataset mini-batch instead of the whole dataset used for calculation.

6.4 Hyper-parameter Tuning

Hyper-parameter tuning involve trial and error approach. But there are few approaches that can be adopted to reduce the time-period to find appropriate hyper-parameters. Randomly test hyper-parameter values from the multi-dimension hyper-parameter plane. Once some regions in this multi-dimensional plane start giving good results, zoom out the region and try more random values from this region to find the most appropriate combination.

Scaling of hyper-parameter random value selection is important to maintain uniformity. By using the logarithmic scale, the value can be randomly uniformly distributed. For the regression problem, if the exponential weighted averaging technique is used, then it can be visualized that randomly selected hyper-parameter value will be close to zero and this result in an over-fit model.

Additionally, multiple models can be trained at a time, and the best performing one will be selected; this is called Caviar approach. Otherwise, in the flow Panda approach, single model is trained with necessary modification in hyperparameter from time to time as per the performance.

6.5 Convolutional Neural Network

Convolutional Neural Network is a deep net similar to the neural net but is not mandatory that all the neuron between subsequent layers will be interconnected for the deep net. It's computationally expensive and time consuming to feed all the image pixels to the deep net. So, instead of feeding the whole image to the deep net, only the most import spatially invariant feature is given as an input to the deep net. CNN has three major stages, convolution, pooling, and flattering.

Convolution is used to extract features from the images using kernels/filters. Equation (44) represent the convolution operation between two matrix.

$$f(x,y)\circ g(x,y)=\sum_{n=0}^{cols}\sum_{m=0}^{row} g(n,m)*f(x-n,y-m) \tag{44}$$

here dot product is between f(x,y) the actual image matrix and g(x,y) the kernel matrix is performed. The kernels are the features filter whose weight has been modified over the training period to detect a specific feature of the image. Some common kernels are sharpened kernel, edge detection kernel, smoothing kernel, etc. The kernel is rolled from left to right and top to bottom and result to feature map. The size of the feature map matrix is less the actual image, so in case of size has to be kept unchanged, then the padding has to be performed on the actual image. Padding involves adding of extra black pixel around the image. Relu operation is performed on the feature map to replace negative pixel value to zero. There can be multiple convolution stages in a model where all the feature map from the previous map is repeatedly convolved with a new set of filter, and then each set is summed individually to generate a new feature map, as shown in Figure 23. Number of generated feature map represent the dept of the layer.

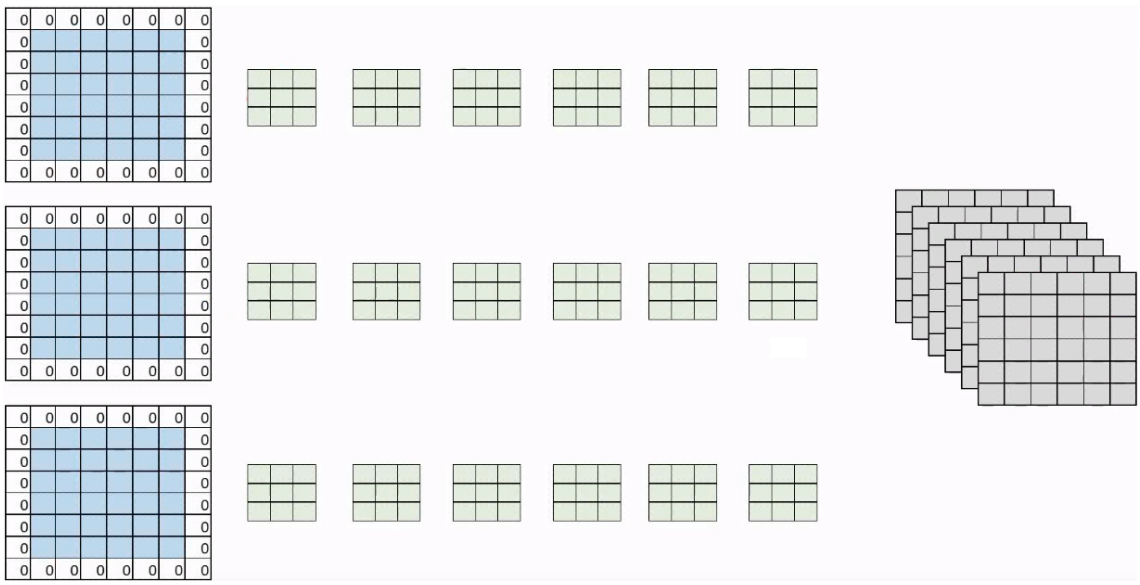

Figure 23. Created feature map by convolving previous layer feature map and sets of new filters.

To make model spatially invariant so the classification can be performed irrespective of the position of the object, pooling is performed. It is performed right to left and top to bottom by shifting pooling matrix size at a time. It specially used to reduce the size of feature matrix by summarizing a region of feature map to a group of pooled matrix values. There can either max pooling, in which the highest pixel value from the selected pool matrix region is considered for pooled matrix, or there can be average pooling in which the average value of the pixels from the selected pool matrix region is considered.

Flattering is performed on the pooled matrix to make it suitable for the deep neural net. This involves turning pooled matrix's to the single-column matrix, and feed to the deep net input layer, as shown in Figure 24. The number of neurons in the subsequent hidden layer signifies the number of filters used over the previous layer and filter matrix value is equivalent to edge-weights, as shown in Figure 25.

During the training, the weights for the filter used along with the convolutional layer and inside deep net will keep getting the update. CNN makes a more generalized understanding of objects in the image and the narrow down to features details of the object to collect concrete evidence about the presence of an object in a flattering matrix compressed form, which is further fed to the deep net for the classification of the object. For transfer learning, the filter weight of the last few convolutional layers and the deep net is reconfigured to retrained the model.

Figure 39. CNN Architectural example.

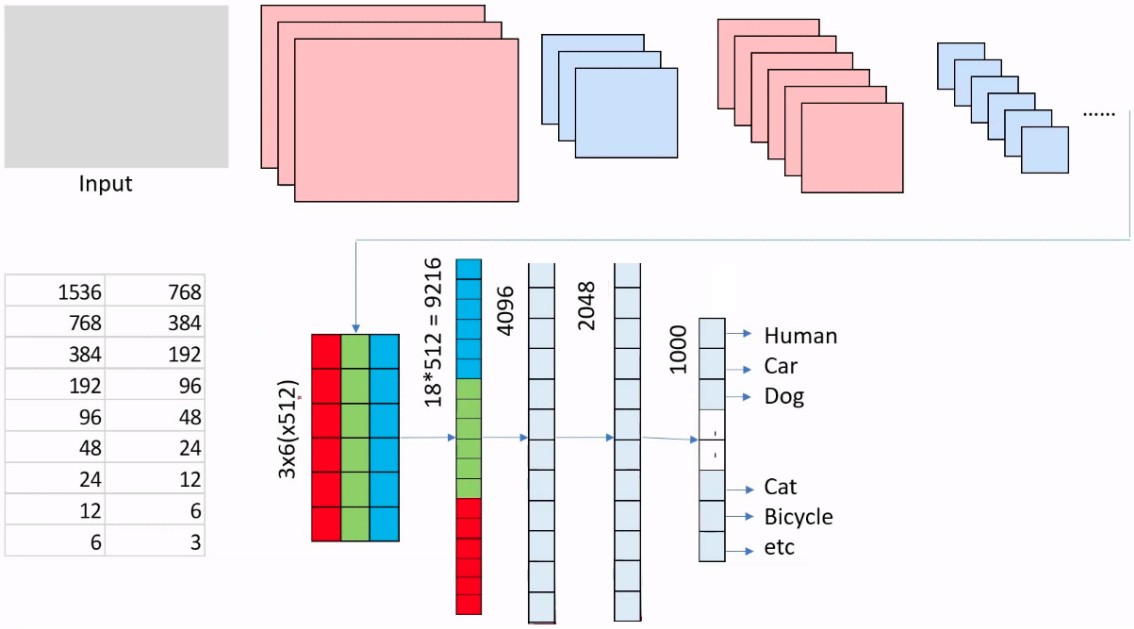

Figure 24. CNN Architectural example.

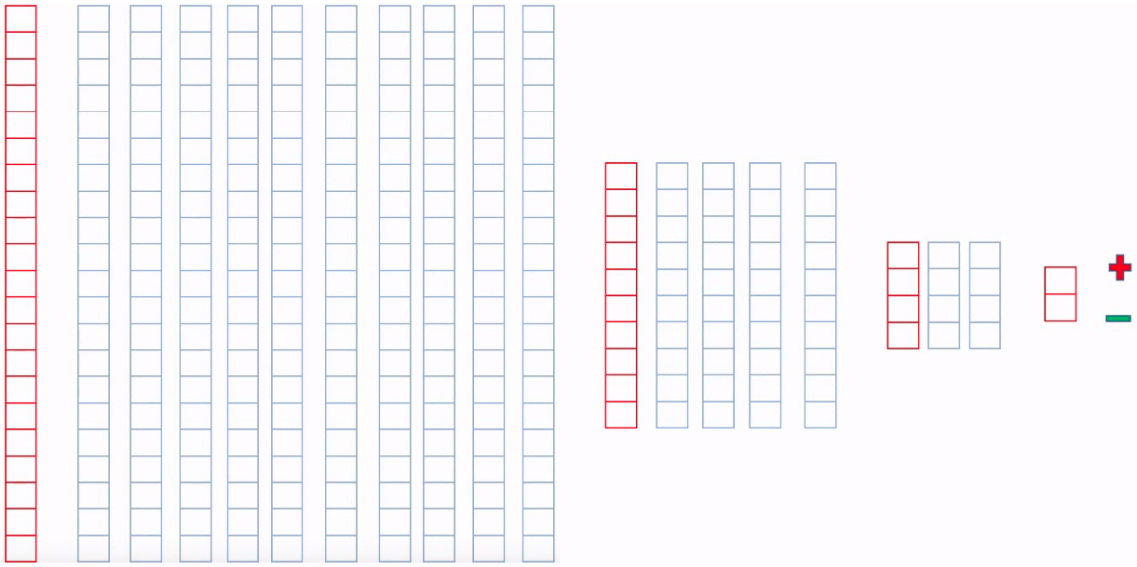

Figure 25. Deep net in CNN.

6.6 Object Detection

Object detection is a combination of object localization and classification. In output, along with looking for the type of object, the prediction is also performed for the location of the object. The label is used to confirm the object type while training at the same time model learns about the size dynamics of the object as per the minor features. The predicted size is compared with the labelled bounding box for the object to adjust the weights. This results in a model that can do classification and localization at a time.

There can be multiple objects of different sizes in an image. To do multiple object detection scrolling window and image-pyramid is used. Every model has a limitation for input image size. The scrolling window involves a bounding box of the size specified by the model for the input image. Bonding box is used to detect multiple objects in a dataset image by scrolling a box from right to left and top to bottom by the multiple selected stride size. Classification and localization are performed on each of the bounding boxes. CNN can do multiple object detection in an image. The same objects can be detected in the multiple bounding boxes. Outside CNN, the confidence score is assigned to each bounding box, which is based on how precisely the object is classified and localized in that bounding box. Then on the bases of the confidence score of the bounding box, which has detected the same objects, will be the object localization bounding box.

In case object size exceed the model input-image limit as well as the bonding box used for that image, then image-pyramid is used to scale down the size of the image and then feed images to the model for classification and detection. The number of input layer neurons remains unchanged irrespective of the different sizes of the same image used for object detection and localization. One problem with different size images is a different size flattered single column matrix. So instead of using the flattered matrix, use the last layer pooled matrix in small sizes as per the count of input layer neuron. This results in a multi-size output matrix for each class. The combination of outputs from different sizes of the same image is used to object detection and localization.

6.7 Region Based CNN (R-CNN) [13]

In CNN, sliding window technique supported for object detection and localization. But this increases the computation with a decrease in stride size. R-CNN uses the “Region-based technique” as a pre-step to avoid sliding window and image pyramid.

Multi-scale Saliency is a region-based technique that using FFT filter to localize the region of interest in the image. If the input image is RGB based, then “Colour Contrast” technique is used, find the region of object-based to intensity of colour. “Superpixels Straddling” suggest region on bases of intensity and sequentially combine regions to localize the object's region. “Edge Detection” is the other technique that selects region with maximum edges as region of interest. Next to “Edge Detection” is “Edge Boxes” which looks for the contour. It uses the random forest to find different perspectives about edges in an image and then combine them to prepare image contour. Then the sliding window is used to find region having well-formed contour, signify the region of interest. “Selective Search” is more an effective approach, it combines region selection based on color, edges, texture, etc. and if parts of the same objects are of a different color, then use image composition to combine them.

Whatever the region-based technique used to find the region of interest, the respective region cropped image is warped to 227x227 size input image for R-CNN. Figure 26 shows the R-CNN architecture. AlexNet/VGG model is developed over the pre-trained ImageNet model. ImageNet is trained on the full-size image and R-CNN uses warped images, fine-tuning of weights in the convolutional layer and the deep net is performed using softmax activation function at the last layer. Secondly, the model is trained with SVM as the last layer activation function. Thirdly, the model is trained for localization. R-CNN transfer learning gives intuition that every layer weight, in combination with last layer activation, improves the accuracy of the model.

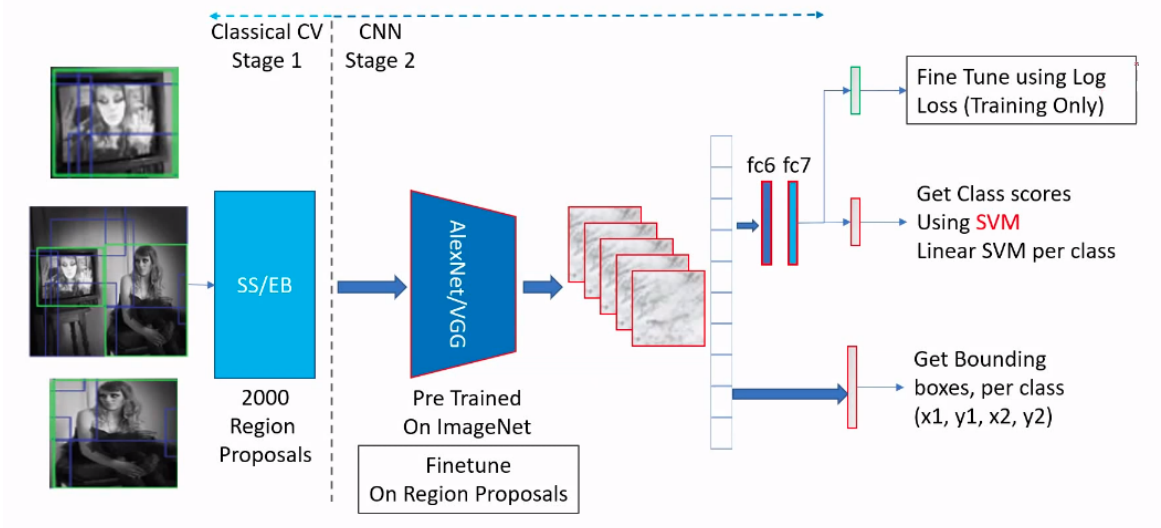

Figure 26. R-CNN Architecture.

R-CNN replace softmax with SVM. Softmax assigns probability density to each classified class object. For the SVM machine during training, multiple dimensional output matrix element will play multiple dimension SVM plane. It's soft SVM, so hing loss is used overtraining to find precise hyperplane between classes. Training also makes SVM

robust to the scenario in which multiple objects of the same of different classes are detected. In real-time, output matrix elements are plotted to find the object class.

Region proposal makes R-CNN slower, but it improves the accuracy of the model by providing the most promising region to detect and localize objects. This significantly reduces the possible false positive from the background region.

6.8 Faster R-CNN

Faster R-CNN is a combination of CNN and “Region Proposal Network (RPN)”. RPN is a neural network of it’s own, which is trained to provide the region of interest.

In RPN, first, the bounding box regression is performed to find the background region and foreground region using non-max pooling. For each foreground bounding box, nine anchor boxes of different sizes and aspect ration are used to detect an object within the box. The first convolutional layer matrix of RPN is of size 3x3, which corresponds to 228x228 image region. The anchor boxes are of area 128 sq unit, 256 sq unit and 512 sq unit with an aspect ration of 1:1, 1:2 and 2:1, as shown in Figure 27. So it’s possible that anchor box is larger or smaller then bounding box in height and width, and not cover complete object. The RPN will use this anchor box information and roughly localize the object in the image, which further fine-tuned by CNN.

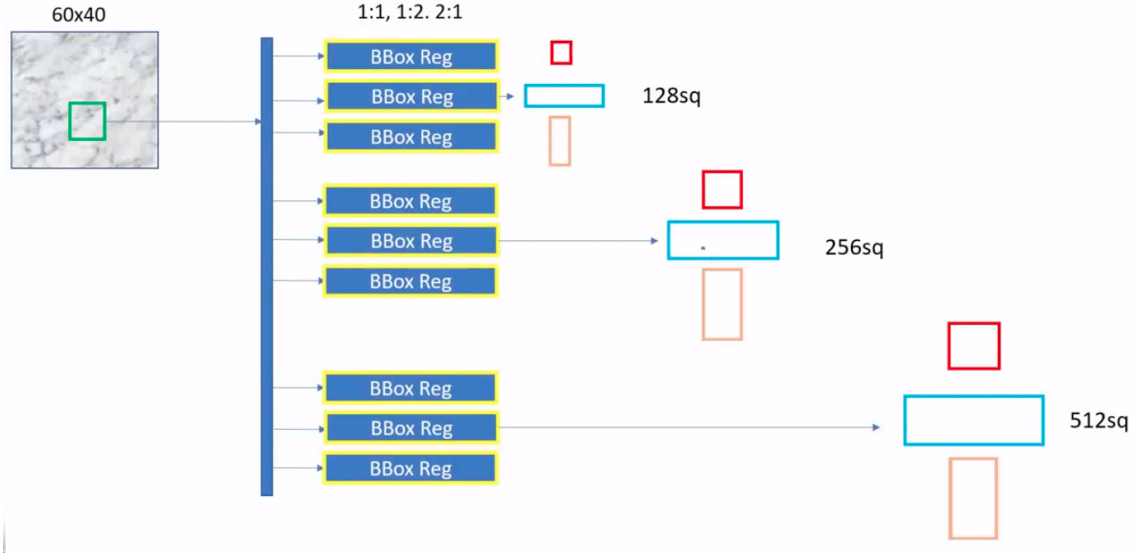

Figure 27. Anchor box in fast R-CNN.

For faster R-CNN, multi-scale max spatial programming pooling on the last layer feature map is replaced with 7x7 max pooling. Classification and localization are combined for training. Classification process "log loss" and Localization process "Smooth L1 loss" is combined and used for back-propagation.

While training RPN and CNN first, RPN ConvNet is trained. Then Fast-R-CNN is trained using it’s ConvNet and RPN proposals. Then to make RPN compatible with CNN feature map, weights for RPN ConvNet are tunned; this results in the changes of ROI proposal by RPN. So to make CNN ConvNet compatible with the new ROI proposal, weight for CNN ConvNet is tuned.

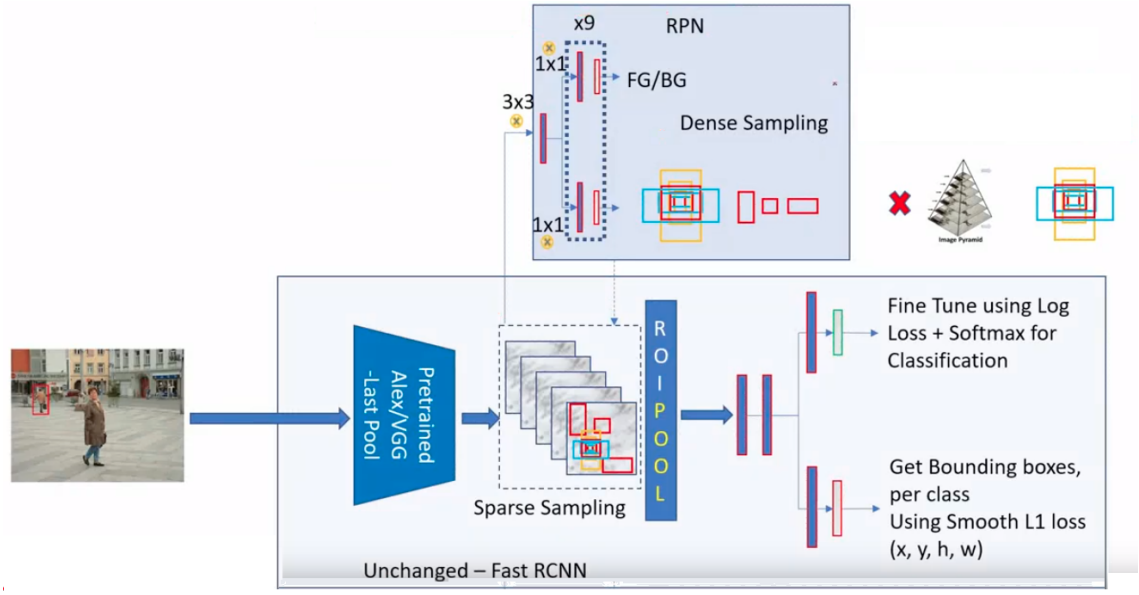

Figure 28. Faster R-CNN Architecture.

Faster R-CNN will predict a lot of bonding boxes for all the classes; to reduce the bounding box, the bounding boxes with top confidence scores are selected, and non-max separation is applied to find the most appropriate box for an object. Figure 28 display faster R-CNN architecture.

6.9 Recurrent neural network (RNN)

RNN is a time series based prediction model. With the feed-forward neural network, the classification happens all at the moment. With RNN, glim's of the input are learned in a sequential manner to do classification. Figure 29 shows the basic architecture of RNN and equation (45), (46) and (47) show the equivalent equation. This is a time-domain, so model skeleton memory is fixed. What can vary is either the input size of output size. Figure 50 shows the iteration for the time-domain RNN model.

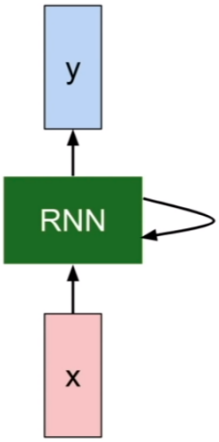

Figure 29. Basic architecture.

$$h_t = f_w(h_{(t-1)}, x_t) \tag{45}$$

$$h_t = \tanh(W_{hh} h_{(t-1)} + W_{xh} x_t) \tag{46}$$

$$y_t = W_{hy} h_t \tag{47}$$

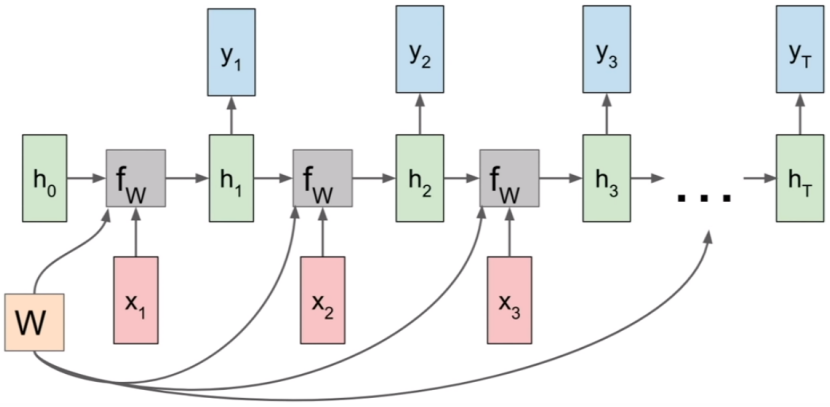

Figure 30. Iteration for time domain RNN model.

In the time domain model, it's computationally expensive to go back in time, so back-propagation is performed for some interval of time in the past. It's a reinforcement learning, and the number of iteration has to be mentioned. The output of each iteration of RNN is a vector as per the number of classes. The output vector is in the spatial domain, so it don't have to consider outputs from the previous iteration.

For object detection, CNN and RNN are combined to predict the context of the image. Each iteration provides location distribution and vocabulary distribution matrix, as shown in Figure 31.

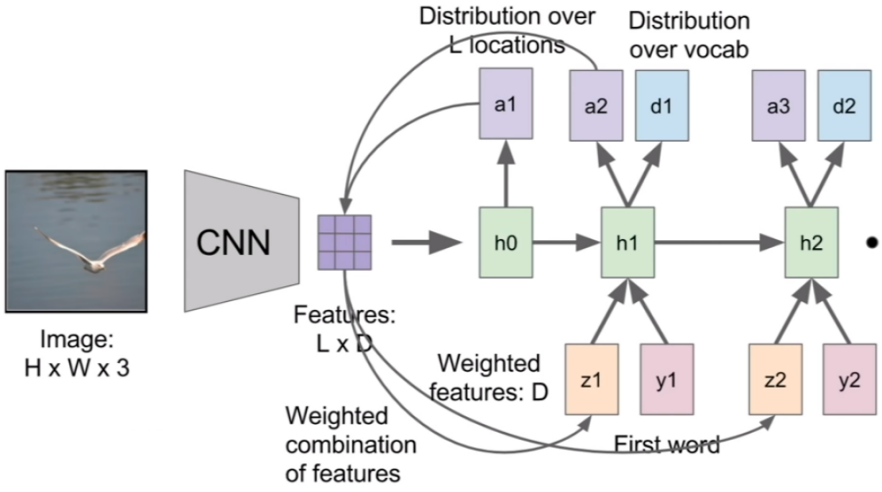

Figure 31. Objection detection with the combination of CNN and RNN.

RNN uses tanh activation function for the hidden layer, whose significance reduce with each derivation. Computing gradient to tanh function can lead to vanishing gradient problem of exploding gradient if the initial value is very large. The exploding gradient can be controlled by clipping of steps during gradient convergence. For vanishing, the gradient LSTM model is used.

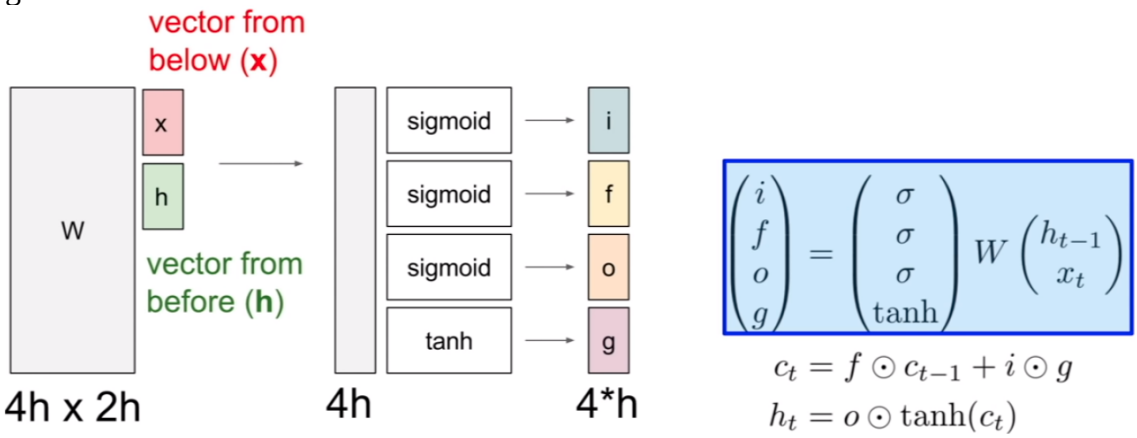

Figure 32. LSTM Model.

For LSTM, hidden and inputs are used to calculate gates value. The cell state and hidden state is computed from the gates. LSTM cell has four gates within; those are Forget gate, Input gate, Gate gate, and Output gate. Forget gate decide whether to erase cell or not. Input gate responsible for deciding writing to the cell. Gate decides how much to write. Output gate control cell output. Figure 32 show typical LSTM model cell. The benefit with LSTM is that while back-propagation, the gradient of tanh is avoided that reduces vanishing gradient problem, as shown in Figure 33.

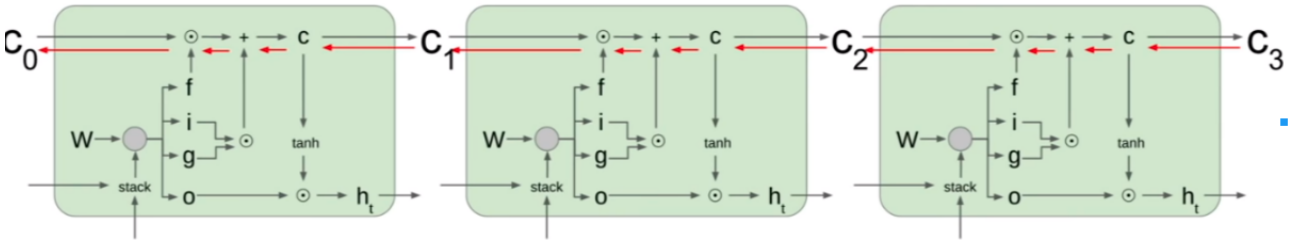

Figure 33. LSTM multiple iteration.

7 Data Acquisition

Data is acquainted from the University of Texas at Arlington Real-Life Drowsiness Dataset (UTA-RLDD) [15] and National Tsing Hua University Cvlab-Driver Drowsiness Detection Dataset. The UTA-RLDD database has videos of alert, neither alert nor drowsy and drowsy videos of 60 participants. In total, there are 180 videos of approximately 10 mins each. I have used only alert and drowsy videos of 16 participants. Frames are being extracted from each video. Each video provides approximately 18000 frames. Only 16 participants are used because of the limitation of the system memory and processing calibre and time constraints for training a model.

The assumption is facial features can be used to deduce information about a person's state of alertness. These facial features include changes in eyes and mouth movements. Closed eye and yawning for a longer duration is the sign of drowsiness. To record close eyes and yawning eye-aspect ratio (EAR) and mouth-aspect ratio (MAR) has to be calculated. Information about the facial landmark is required to calculate EAR and MAR. Dlib library provides a pre-trained facial landmark detector to detect 68 landmarks on the face with x and y coordinates, as shown in Figure 34. Equation (48) and (49) show EAR and MAR calculation. Out of 68, 14 landmark coordinate is given as the input to the model.

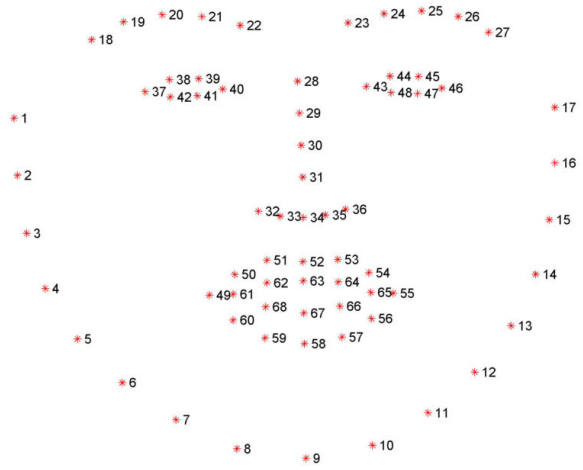

Figure 34. Facial landmarks provided by dlib library.

$$EAR = \frac{\|P_{38} - P_{42}\| + \|P_{39} - P_{41}\|}{2\|P_{37} - P_{40}\|} \tag{48}$$

$$MAR = \frac{\|P_{62} - P_{68}\| + \|P_{63} - P_{67}\| + \|P_{64} - P_{66}\|}{2\|P_{61} - P_{65}\|} \tag{49}$$

I used OpenCV to read frames from videos. The BGR color images are converted to grayscale images to save the model from learning unwanted color objects in the image. Dlib face landmark model provides 68 landmark coordinate. Landmark coordinates from each frame of the video are saved to csv file. Each video information is saved in separate text files. The extracted feature information is inputted to the model for training purposes. Appendix 1 shows the program code for data acquisition. The Table 1 and Table 2 show the system and library specification used for programming.

Table 1. System specification used for training

OS	Linux – Ubuntu 16.04
Kernel	4.15.0-96-generic
Processor	Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
Processor Aricheture	64 bit architecture
RAM	15GB

Table 2. Library information used for programming

Library	**Version**
Python	2.7.12
OpenCV	2.3.9.1
Pandas	0.24.2
Imutils	0.5.2
Numpy	1.16.2
Keras	2.2.4
Tensorflow	1.13.1
Matplotlib	2.2.5
Sklearn	0.20.4

8 Modelling

The extracted feature information is feed to the model. All the video csv file is read and transformed to panda dataset table. Eyes and mouth aspect ratio is calculated using the table. The 2-dimensional dataset is converted to a 3-dimensional table, which is inputted to LSTM model. The whole data set is divided into 74:26 training by testing ratio. The ratio is selected in such a way that for transforming dataset to 3-dimension, there can be a common factor of training and testing dataset length. “16” is the common factor for the training and testing dataset length.

The paper “*Driver drowsiness detection using behavioural measures and machine learning techniques: A review of state-of-art techniques*"[3] has used a 3D-CNN, by considering time as a dimension apart from frame dimensions. This inspired to experiment and evaluate 3D model. 3D-CNN, or also be called RNN have vanishing gradient problem with the iterative derivation of tanh activation function. LSTM overcome vanishing gradient problem and use historical data frames to predict the status of drowsiness. Predicting drowsiness on the individual frame will generate a huge amount of false-positive due to conscious eye blinking. That is why I decided to use LSTM. The paper “*A neural-network-based investigation of eye-related movements for accurate drowsiness estimation*"[2] have evaluated LSTM using eye data only. I have also include mouth aspect ratio in data set, while evaluating LSTM performance. Appendix 1 shows the program snippet to convert video frame information to panda dataset table, eye and mouth aspect ratio calculation and dataset conversion to three dimensions. The sequential model has an input layer of LSTM network, followed by LSTM hidden layer, following 0.2 dropout layer and softmax activation at the output layer, as shown in Figure 35. Dropout layer is used to overcome the overfitting problem.

K-fold cross-validation is used with 10 folds to find the mean accuracy of the model and to use the most precisely trained model.

```
Loaded model from disk
Learning rate : 0.001 beta1 : 0.9  beta2 : 0.999
Model: "sequential_113"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_225 (LSTM)              (None, 16, 100)           60800
_________________________________________________________________
lstm_226 (LSTM)              (None, 100)               80400
_________________________________________________________________
dropout_112 (Dropout)        (None, 100)               0
_________________________________________________________________
dense_111 (Dense)            (None, 2)                 202
=================================================================
Total params: 141,402
Trainable params: 141,402
Non-trainable params: 0
_________________________________________________________________
```

Figure 35. Model Visualization.

9 Result

The model is trained 10 times with 1000 epochs each and with a learning rate of 0.001. The input layer has 100 input nodes. The accuracy of 10 runs are 79.47019934654236, 100.0, 97.57174253463745, 32.45033025741577, 84.32670831680298, 34.21633541584015, 94.0397322177887, 31.78808093070984, 61.06194853782654, and12.389380484819412. The average accuracy for 10 folds is 62.73 % accuracy. The best-performed model, with an accuracy of 97.57 %, is used for testing. Figure 36 and Figure 37 respectively show the accuracy of the loss curb of the model while training. Figure 38 shows the AUCROC characteristics. AUC turns out to be 0.6; this signifies the model is capable of making more of the right prediction rater then false positive or false negative prediction. The prediction is Figure 39 shows the comparison between ground truth and model prediction. The model prediction doesn't match exactly with the theoretical expectation; the same is shown by Table 3 confusion matrix. True count is more then false count but isn't significant enough.

Table 3. Confusion matrix for the model prediction.

N = 1591	Prediction: **Drowsey**	Prediction: **Alert**	
Acutal: **Drowsey**	659 (**TP**)	274 (**FP**)	933
Acutal: **Alert**	314 (**FN**)	344 (**TN**)	658
	973	618	1591

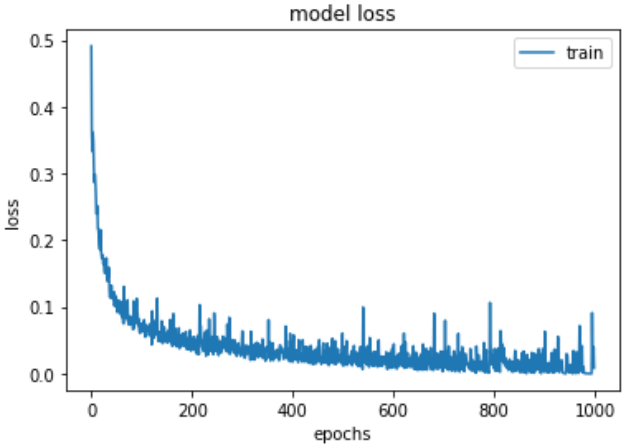

Figure 36. Loss curve for 1000 epochs.

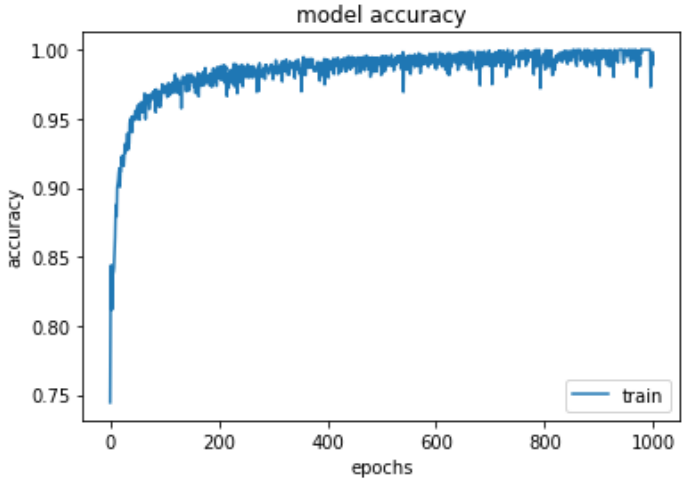

Figure 37. Accuracy curve for 1000 epochs.

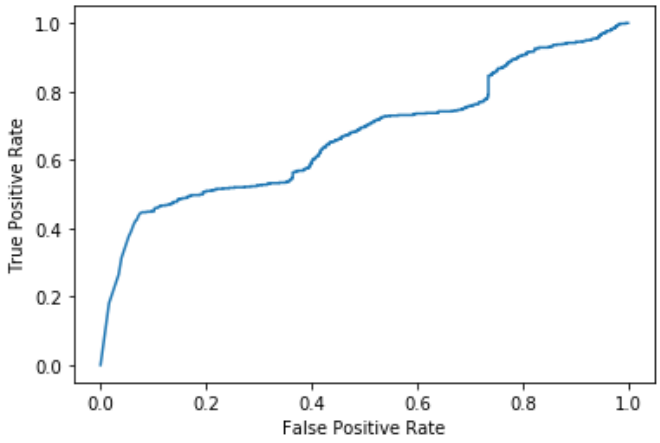

Figure 38. AUCROC characteristics of model.

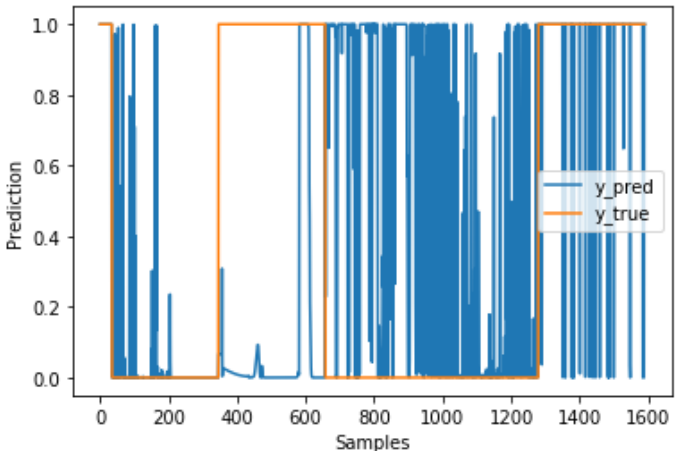

Figure 39. Ground truth and model prediction

10 Summary

The literature review help to find an approach for this work. Theory section of this work padded with implementation knowledge of different possible models, cost functions, activation function, evaluation techniques. As per the result, this can be concluded that the approach of the 3-dimensional LSTM model is not enough to make a reliable prediction for drowsiness detection. The result shown in Figure xx is expected by me to some extent because, in dataset videos of 10 minutes, the participant isn't drowsy throughout the video; still, the complete videos are labelled as drowsiness state of the participant. This way, the model intentionally got trained to generate false negatives. LSTM consider historical data for prediction because of the previous false negative influenced the present prediction. The dataset isn't good enough because, the 10 minutes long videos aren't very useful as the video duration during which participant is actually feeling drowsy are significantly small. The shorter videos of deep drowsiness state will be a more favourable candidate for a better performing model.

Table 4. Performance matrix

Performance Fields	Value	Formula
Accuracy	0.63	(TP+TN)/ Total
Miscalssification rate	0.369	(FP+FN)/ Total
True positive rate	0.706	TP/ Actual drowsey
False positive rate	0.416	FP/ Acutal alert
True negative rate	0.522	TN/ Actual alert
Precision	0.677	TP/ Predicted drowsey
Prevalence	0.586	Actual drowsey/ Total

One of the major pieces of the objective to do face detection on the bases of the facial landmark is successful. Table 4 shows the model performance, numerical summary on the test dataset. The accuracy of 0.63 and true positive rate of 0.7 are not satisfactory. As per the paper "*Driver drowsiness detection using behavioural measures and machine learning techniques: A review of state-of-art techniques*"[3] , the author mentioned CNN models, which manage to achieve accuracy between 0.83 to 0.98. The result shown on the test dataset in Figure 39 isn't fully reliable. The prediction is made for every 100ms

video period. On checking the tested video manually, it has observed that the prediction is actually correct, even the whole video is labelled as the only drowsy or alert state.

From a realistic point of view, the facial landmark data is not enough to make a reliable prediction because of the illumination condition inside the vehicle and the use of opaque or translucent objects to cover parts of the face. Information about participant body temperature and heart rate can definitely improve true-positives. The intrusive approach using wearable can be used to collect body temperature and heart rate.

LSTM accuracy signifies that this approach has good potential for drowsiness detection with some improvement. The future work will be to train the same model with shorter videos, having a deep drowsiness state of participant, and also include heart rate and body temperature data to model data set. Port whole of the system to the embedded board having a neural unit or adding neural unit peripheral for machine learning calculations. Deploy system on edge.

References

[1] R. Manoharan; S. Chandrakala , "Android OpenCV based effective driver fatigue and distraction monitoring system" in 2015 International Conference on Computing and Communications Technologies (ICCCT).

[2] Mingfei Sun; Masanori Tsujikawa; Yoshifumi Onishi; Xiaojuan Ma; Atsushi Nishino; Satoshi Hashimoto, "A neural-network-based investigation of eye-related movements for accurate drowsiness estimation" in 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).

[3] Mkhuseli Ngxande; Jules-Raymond Tapamo; Michael Burke, "Driver drowsiness detection using behavioural measures and machine learning techniques: A review of state-of-art techniques" in 2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech).

[4] Kartik Dwivedi; Kumar Biswaranjan; Amit Sethi, "Drowsy Driver Detection using Representation Learning" in 2014 IEEE International Advance Computing Conference (IACC).

[5] Bhargava Reddy; Ye-Hoon Kim; Sojung Yun; Chanwon Seo; Junik 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).

[6] Ashish Kumar; Rusha Patra, "Driver Drowsiness Monitoring System using Visual behaviour and Machine Learning", in 2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE).

[7] Charlotte Jacobé de Naurois; Christophe Bourdina; Anca Stratulat; Emmanuelle Diaz; Jean-Louis Verchera, "Detection and prediction of driver drowsiness using artificial neural network models", Volume 126, May2019, in 10th International Conference on Managing Fatigue: Managing Fatigue to Improve Safety, Wellness, and Effectiveness.

[8] Sean Harrington, "Solving Logistic Regression with Newton's Method", July 2017, on Math-of-machine-learningm,[Online]. Available: https://thelaziestprogrammer.com/sharrington/math-of-machine-learning/solving-logreg-newtons-method.

[9] "Understanding the Bias-Variance Tradeoff", June 2012. [Online]. Available: http://scott.fortmann-roe.com/docs/biasvariance.html.

[10] "Covariance Matrices and Data Distributions", March 2019, [Online]. Available: https://theclevermachine.wordpress.com/2013/03/29/covariance-matrices-and-data-distributions/.

[11] Michael Nielsen, "Neural Networks and Deep Learning is a free online book", Chapter 3, Dec 2019, [Online]. Available: http://neuralnetworksanddeeplearning.com/chap3.html#regularization.

[12] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun , "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks", 21 Dec 2013 , Computer Vision and Pattern Recognition course by Cornell University.

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, "Region-based Convolutional Networks for Accurate Object Detection and Segmentation" in IEEE Transactions on Pattern Analysis and Machine Intelligence.

[14] Ng, Andrew, Coures CS229 - Machine Learning, Standford University. [Online]. Available: https://see.stanford.edu/Course/CS229

[15] UTA Real life data set. [Online]. Available: https://sites.google.com/view/utarldd/home

[16] Driver Drowsiness Detection Dataset. [Online]. Available: http://cv.cs.nthu.edu.tw/php/callforpaper/datasets/DDD/

[17] United state department of transportation. [Online]. Available: https://www.nhtsa.gov/risky-driving/drowsy-driving

Appendix 1 – Program Snippet

Program code for data acquistion.

```
def process(frame):
    return_list = []
    # Frames read from video
    img = frame

    # BGR color image converted to Gray scale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # DLib2
    rects = detector(gray, 0)
    if len(rects) == 0:
        return_list = ['-1' for x in range(4+(68*2))]
    if len(rects) > 1 :
        rects = [rects[0]]

    for (x,y,w,h) in [face_utils.rect_to_bb(x) for x in rects]:
        return_list += [x, y, w, h]

        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

        shape_img = dlib.rectangle(int(x), int(y), int(x+w), int(y+h))
        shape = predictor(img, shape_img)
        shape = face_utils.shape_to_np(shape)

        # Facial land mark displayed over face
        if len(shape) ==0:
            return_list += ['-1' for x in range(68 * 2)]
        else:
            assert(len(shape) == 68)
            return_list += [item for sublist in shape.tolist() for
item in sublist]
            for (xx, yy) in shape:
                cv2.circle(img, (xx, yy), 1, (0, 0, 255), -1)

    # Display an Image
    cv2.imshow('img',img)
    cv2.imshow('gray cut',roi_gray)
    if not (len(return_list) == 68 * 2 + 4):
        print('len :', len(return_list))
        print('return_list :', return_list)
    return return_list
```

Program code for covert video frame information to panda dataset table.

```
def get_table(participant, mood, start_time=61, stop_time=559,
resample_interval='10000ms',
                base_path=None):

    # Look for File
    if base_path is None:
        base = os.path.join('output','csv')
    else:
        base = base_path

    files = glob.glob(os.path.join(base, str(participant)
+'_'+str(mood)+'.csv'))

    if(len(files) !=1 ):
        logging.error("Looked for "+str(participant)+'_'+str(mood)
+'.csv and found '+str(len(files))+' tables. Need to match with one
table only.')
        raise RuntimeError

    # Load File
    logging.info("Loading "+str(files[0]))
    table = pd.read_csv(files[0])

    # Resample Time
    table['date'] = pd.to_datetime(table.time, unit='s')
    if resample_interval is not None:
        table = table.resample(resample_interval, on = 'date').mean()
    else:
        table.set_index('date', inplace = True)

    # Drop unwated columns
    table = table.filter(columns_to_keep)

    # Trim intial and end of video
    table.drop(table[ table['time'] > stop_time ].index, inplace=True)
    table.drop(table[ table['time'] < start_time ].index,
inplace=True)

    # Fill NaN data
    table.replace(-1, np.NaN, inplace=True)
    table.interpolate(inplace=True, limit_direction='both')

    # Set Data Types
    table[['participant', 'mood']] = table[['participant',
'mood']].astype('int32')
    return table
```

Program code to calculate eye and mouth aspect ration.

```
def ratio_6(table, t1,t2,b1,b2,l,r):
    x1_m= mid(table['px_'+str(t1)], table['px_'+str(t2)])
    y1_m = mid(table['py_'+str(t1)], table['py_'+str(t2)])
    x2_m = mid(table['px_'+str(b1)], table['px_'+str(b2)])
    y2_m = mid(table['py_'+str(b1)], table['py_'+str(b2)])

    return dist(x1_m,y1_m,x2_m,y2_m) / dist(table['px_'+str(l)],
table['py_'+str(l)], table['px_'+str(r)], table['py_'+str(r)])

def ratio_4(table, t,b,l,r):
    return
dist(table['px_'+str(t)],table['py_'+str(t)],table['px_'+str(b)],table
['py_'+str(b)]) / dist(table['px_'+str(l)], table['py_'+str(l)],
table['px_'+str(r)], table['py_'+str(r)])
```

Program code to train-test split and coverting datset to 3 dimesnion.

```
dataset4train_size = int(dataset.shape[0] * train_test_split)

X_train     = X[0:dataset4train_size]
X_test      = X[dataset4train_size:X.shape[0]]
y_train     = y[0:dataset4train_size]
y_test      = y[dataset4train_size:X.shape[0]]

# Reshape input to 3D
train_resize    = int(X_train.shape[0]/n_steps)
test_resize     = int(X_test.shape[0]/n_steps)

X_train     = X_train.reshape(train_resize, n_steps, X.shape[1])
y_train     = y_train[::n_steps]
X_test      = X_test.reshape(test_resize, n_steps, X.shape[1])
y_test      = y_test[::n_steps]
```

Program code for LSTM model with 10 K-fold.

```
for train_index, test_index in KFold(n_split).split(X_train):
    x_4train, x_4valid = X_train[train_index], X_train[test_index]
    y_4train, y_4valid = y_train[train_index], y_train[test_index]
    # Create model
    model = Sequential()
    model.add(LSTM(100, input_shape=(x_4train.shape[1],
x_4train.shape[2]),return_sequences=True))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(2,activation='softmax'))
```

```
    # Compile model
    model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=['accuracy'])
    print("Learning rate :", K.eval(model.optimizer.lr), "beta1 :",
K.eval(model.optimizer.beta_1), " beta2 :",
K.eval(model.optimizer.beta_2))
    # Fit model
    start = timeit.default_timer()
    history = model.fit(x_4train, y_4train, epochs=1000,
batch_size=32, verbose=1)
    stop = timeit.default_timer()
    print('Time: ', stop - start)

    histories.append(history)
    # Evaluate the model
    scores = model.evaluate(x_4valid, y_4valid, verbose=1)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)

    # summarize history for loss
    pyplot.plot(history.history['loss'])
    #pyplot.plot(history.history['val_loss'])
    pyplot.title('model loss')
    pyplot.ylabel('loss')
    pyplot.xlabel('epoch')
    pyplot.legend(['train', 'test'], loc='upper left')
    pyplot.show()

    # summarize history for accuracy
    pyplot.plot(history.history['accuracy'])
    #pyplot.plot(history.history['val_accuracy'])
    pyplot.title('model accuracy')
    pyplot.ylabel('accuracy')
    pyplot.xlabel('epoch')
    pyplot.legend(['train', 'test'], loc='upper left')
    pyplot.show()
    # Serialize model to JSON
    model_json = model.to_json()
    with open("model/fold"+str(fold)+"_model.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model/fold"+str(fold)+"_model.h5")
    print("Saved model to disk")
    fold += 1
```