

https://dspace.org/

Real-time vision-based driver alertness monitoring using deep neural network architectures

Olamijuwon, Olugbenga Jeremiah

2020
https://hdl.handle.net/10539/31089

Downloaded from a DSpace repository. https://dspace.org/

UNIVERSITY OF THE WITWATERSRAND



WITS
UNIVERSITY

School of Computer Science and Applied Mathematics
Faculty of Science
MASTERS THESIS

Real-time Vision-based Driver Alertness Monitoring using Deep Neural Network Architectures

Olugbenga Jeremiah Olamijuwon
(Student Number: 1744903)

Supervisor: Professor Turgay Celik

April 20, 2020, Johannesburg

Declaration

I, **Olugbenga Jeremiah Olamijuwon**, declare that ALL the work submitted for assessment of this dissertation titled **"Real-time Vision-based Driver Alertness Monitoring using Deep Neural Network Architectures"** is my own unaided work except where I have explicitly stated otherwise, and no portion of this dissertation has been submitted at this or any other institution of learning. I have also followed the required conventions in referencing the thoughts and ideas of others.

Signature: 

Signed on August 18, 2020 in Johannesburg.

Abstract

According to World Health Organization, driver drowsiness and distraction are leading factors of road crashes. In 2013, approximately 1.25 million deaths were caused by road traffic injuries [1] and by 2018, this number had swelled up to 1.35 million [2]. This report can be interpreted to mean that approximately one person experiences (or will experience) a fatal crash every twenty four seconds [3] [2]. Thus, because of the rising drowsiness related accidents, it is pertinent to know the state of a driver in real time to allow for a prior warning to either take a break or hand over the steering wheel to another person (or an artificial intelligent smart vehicle where applicable). To achieve this goal, there is a need to develop a system which continuously monitors the driver's alertness in real time. In this research, we introduce a non-intrusive novel approach to monitor driver alertness in real time using deep neural network architectures. Specifically, we used deep convolutional neural network architectures to design a system which monitors driver alertness in real time. Convolutional neural networks (CNNs) have been notably known as the best machine learning architectures when it comes to image classification. Thus, in this research, we explored different convolutional neural network architectures in developing an efficient face-state detection model. The decisions of these CNNs are fed to our drowsiness evaluator which is formed with conditions based on our findings from the literature. Specifically, we focused on LeNet-5 and four other notable convolutional neural network architectures which have won (or performed excellently) at the ImageNet challenge and these include AlexNet, Inception-ResNet V2, VGGNet ,and ResNext-50. The neural networks were trained on publicly available dataset as well as publicly available images we compiled using Google and Bing APIs. We compared the performance of our proposed driver alertness monitoring framework using the different CNNs against one another in terms of their accuracy, computation speed, Video Error and Average Video Error. Experimental results show ResNext-50 to be our best performing model, even though it only had the second best Average Video Error, after Inception ResNet V2, by a gap of 0.006. In previous literature, researchers have focused on predicting drowsiness with single frames classification, however, this is not an efficient approach as it raises a lot of false positives and is not robust to other drowsiness conditions. Furthermore,previous literatures focus on drowsiness classification and not detection, however, in this study, we have presented a framework for detecting and monitoring drowsiness, and categorizing alertness into different levels which are: Alert, Slightly Drowsy, Moderately Drowsy, Very Drowsy, and Dangerously Drowsy.

Acknowledgements

My heartfelt gratitude goes to Prof. Turgay Celik for the counsel and guidance thus far in this research, for the time and efforts dedicated to my research. His support, counsel and guidance towards this research is highly encouraging and invaluable. I would like to appreciate E-blocks Software and Wits Financial Aid Office for the financial support provided during this research. Lastly, I say a "big thank you" to my friends and colleagues on Wits Mathematical Science Labs 3rd floor and family members (especially my brother, Emmanuel and my fiancee, Elizabeth) for the moral support provide during the course of this research.

Contents

Declaration i
Abstract ii
Acknowledgements iii

List of Figures **vii**

List of Tables **viii**

1 Introduction **1**
1.1 Motivation 1
1.2 Thesis 3
1.3 Aim 3
1.4 Objectives 4
1.5 Research Questions 4
1.6 Research Limitations 4
1.7 Dissertation Structure 5

2 Background and Related Work **6**
2.1 Introduction 6
2.2 Machine Learning 6
2.3 Traditional Machine Learning 7
2.3.1 Related Works 7
2.4 Deep Learning 8
2.4.1 Related Works 9
2.5 Comparison of Deep Machine Learning and Traditional Machine learning 10
2.6 Conclusion 11

3 Deep Convolutional Neural Networks **12**
3.1 Introduction 12
3.2 Convolutional Neural Network Architectures 19
3.2.1 LeNet-5 19
3.2.2 AlexNet 21
3.2.3 GoogLeNet/Inception 22
3.2.3.1 Inception ResNet V2 26
3.2.4 VGGNet 27
3.2.4.1 VGG-16 30
3.2.5 ResNet 31
3.2.5.1 ResNext-50 35
3.3 Face Detection and Alignment using Multi Task Cascaded Convolutional Neural Network . . . 37

3.4 Conclusion . . . 39

4 Research Methodology **40**
4.1 Introduction . . . 40
4.2 Overall Framework of the Driver Alertness Monitoring . . . 40
4.3 Data Collection . . . 41
4.3.1 Images . . . 41
4.3.1.1 Closed Eyes in the Wild (CEW) . . . 42
4.3.1.2 Self Compiled Dataset . . . 42
4.3.2 Videos . . . 43
4.4 Data Preprocessing . . . 45
4.4.1 Face Detection/Extraction . . . 45
4.4.1.1 Face Detection Using Haar Cascade (OpenCV) . . . 45
4.4.1.2 Face Detection Using Deep Neural Networks (OpenCV) . . . 46
4.4.1.3 Face Detection Using Histogram of Oriented Gradients (Dlib) . . . 46
4.4.1.4 Face Detection Using Deep Convolutional Neural Networks (Dlib) . . . 46
4.4.1.5 Face Detection Using Multi Task Convolutional Neural Networks (MTCNN) 46
4.4.1.6 Selection of Face Detection Algorithms . . . 47
4.4.2 Data Augmentation . . . 48
4.4.2.1 Vertical Shift Augmentation . . . 48
4.4.2.2 Horizontal Shift Augmentation . . . 48
4.4.2.3 Random Brightness Augmentation . . . 50
4.4.2.4 Random Rotation Augmentation . . . 50
4.4.2.5 Horizontal Flip Augmentation . . . 51
4.4.2.6 Vertical Flip Augmentation . . . 51
4.4.2.7 Random Zoom Augmentation . . . 52
4.5 Training . . . 52
4.5.1 Architecture . . . 52
4.5.1.1 LeNet-5 . . . 52
4.5.1.2 AlexNet . . . 53
4.5.1.3 VGG-16 . . . 53
4.5.1.4 Inception ResNet V2 . . . 53
4.5.1.5 Residual Network (ResNext-50) . . . 54
4.5.2 Hyperparameter Tuning . . . 54
4.5.2.1 Epochs . . . 54
4.5.2.2 Batch Size . . . 54
4.5.2.3 Iteration . . . 55
4.5.2.4 Learning Rate (LR) . . . 55
4.5.2.5 Pooling Type . . . 55
4.5.2.6 Weight Initialization . . . 55
4.5.2.7 Activation Function . . . 56
4.5.2.8 Optimizer . . . 56
4.5.2.9 Early Stopping . . . 56
4.5.2.10 Cost Function . . . 56
4.5.2.11 Dropout . . . 56
4.5.2.12 Batch Normalization . . . 57
4.6 Metrics . . . 57

4.6.1 Confusion Matrix (CM) . . . 57
4.6.2 Accuracy . . . 58
4.6.3 Precision . . . 59
4.6.4 Recall . . . 59
4.6.5 F1 Score . . . 59
4.6.6 Speed . . . 59
4.6.7 Video Error (VE) and Average Video Error (AVE) . . . 60
4.7 System Details . . . 60
4.8 Conclusion . . . 61

5 Results and Discussions **62**
5.1 Introduction . . . 62
5.2 LeNet-5 . . . 62
5.2.1 LeNet-5 without Batch Normalization, Pooling type: Maxpooling . . . 62
5.2.2 LeNet-5 without Batch Normalization, Pooling type: Average Pooling (Avgpooling) . . . 63
5.2.3 LeNet-5 with Batch Normalization before Activation . . . 63
5.2.4 LeNet-5 with Batch Normalization after Activation . . . 67
5.3 AlexNet . . . 69
5.4 VGG-16 . . . 69
5.5 Inception-ResNet V2 . . . 72
5.6 ResNext-50 . . . 75
5.7 Observations . . . 75
5.8 Drowsiness Level Evaluator . . . 76
5.9 Evaluation of Architecture . . . 77
5.9.1 LeNet-5 . . . 78
5.9.2 AlexNet . . . 78
5.9.3 VGG-16 . . . 78
5.9.4 Inception-ResNet V2 . . . 78
5.9.5 ResNext-50 . . . 78
5.10 Model Selection . . . 79
5.11 Conclusion . . . 79

6 Discussion, Further Studies and Conclusion **81**
6.1 Challenges . . . 81
6.1.1 Dataset . . . 81
6.1.2 Speed of Processing . . . 81
6.1.3 Time and Device Constraints . . . 82
6.1.4 Drowsiness Conditions . . . 82
6.2 Further studies and recommendation . . . 82
6.2.1 Sequence Prediction . . . 82
6.2.2 Time to Drowsiness . . . 83
6.3 Packaged Model and Code . . . 83
6.4 Conclusion . . . 83

Bibliography **85**

List of Figures

3.1 A simple convolutional neural network architecture consisting of five layers which are: 1.) Input Layer, 2.) Convolution Layer, 3.) Pooling Layer, 4.) Fully Connected Layer and 5.) Output Layer. . . . 14
3.2 Parameter sharing in convolution layers . . . 14
3.3 Local connectivity in convolution layers . . . 15
3.4 Activation map . . . 15
3.5 Commonly used activation functions . . . 17
3.6 Convolution, strides and zero padding . . . 18
3.7 An example of maxpooling using a "patch of size 2×2" at stride of 2. . . . 19
3.8 LeNet architecture by Yann Lecun et al., (1998) [4] . . . 19
3.9 AlexNet architecture by Alex Krizhevsky et al., (2012). [5] . . . 21
3.10 An example of a basic inception module. [6] . . . 23
3.11 Inception module with dimensionality reduction. [6] . . . 23
3.12 GoogLeNet (Inception) architecture used for object classification. [6] . . . 24
3.13 Revised GoogLeNet architecture) [7] . . . 27
3.14 Inception modules in Inception ResNet V2 . . . 28
3.15 Reduction blocks in Inception ResNet V2 . . . 28
3.16 Schema for Inception ResNet V2 [8] . . . 29
3.17 VGG configurations. The depth of the configurations increases (with increase in layers) from A to E) [9] . . . 29
3.18 Residual learning: a building block. [10] . . . 31
3.19 Residual blocks for the native and revised networks.) [10] . . . 32
3.21 Native residual unit and revised residual unit) [10] . . . 35
3.20 Comparing example residual network architecture for ImageNet. The left architecture is a VGG-19 model, middle architecture is a 34 parameter layer plain network and the right architecture is a 34 parameter layer residual network.) [11] . . . 36
3.22 Left: Building block of ResNet; Right: Building block of ResNext with a cardinality of 32) [12] 37
3.23 **(a)**: Aggregated residual transformation (same figure as 3.22). **(b)**: Building block equivalent to (a), implemented as early concatenation, **(c)**: Building block equivalent to (a) and (b), implemented as grouped convolutions. The bold part in (b) and (c) shows the reformulation changes. [12] . . . 37
3.24 MTCNN architectural framework. . . . 39

4.1 Overall architecture of the driver alertness monitoring system. . . . 41
4.2 Closed Eyes in the Wild image samples. . . . 42
4.3 Samples of faces of images scraped from Google and Bing . . . 43
4.4 YawDD - Yawning Detection Dataset . . . 44
4.5 Selected Face detectors' Outputs. . . . 48

4.6 Vertical Shift Augmentation . . . 49
4.7 Horizontal Shift Augmentation . . . 49
4.8 Random Brightness Augmentation . . . 50
4.9 Random Rotation Augmentation . . . 50
4.10 Horizontal Flip Augmentation . . . 51
4.11 Vertical Flip Augmentation . . . 51
4.12 Random Zoom Augmentation . . . 52

5.1 Performance plot of LeNet-5 using maxpooling without batch normalization during training . 64
5.2 Confusion Matrix of LeNet model trained without Batch Normalization using Maxpooling . . 65
5.3 Performance plot of LeNet-5 without batch normalization during training . . . 66
5.4 Confusion Matrix of LeNet model trained without Batch Normalization using Avgpooling . . 67
5.5 Performance plot of LeNet-5 with batch normalization before activation at different learning rates. . . . 68
5.6 Confusion Matrix of LeNet model trained with Batch Normalization before activation using Avgpooling . . . 69
5.7 Performance plot of LeNet-5 with batch normalization after activation at different learning rates. 70
5.8 Confusion Matrix of LeNet model trained with Batch Normalization after activation using Avgpooling . . . 71
5.9 AlexNet performance plot during training . . . 71
5.10 Confusion Matrix of AlexNet model trained with batch normalization . . . 72
5.11 VGG-16 performance plot during training . . . 73
5.12 Confusion Matrix of VGG-16 model . . . 73
5.13 Inception ResNet V2 performance plot during training . . . 74
5.14 Confusion Matrix of Inception-ResNet V2 model . . . 74
5.15 ResNext-50 performance plot during training . . . 75
5.16 Confusion Matrix of ResNext-50 model . . . 76

List of Tables

3.1 Description of LeNet-5 architecture . . . 20
3.2 Description of AlexNet architecture . . . 22
3.3 Description of GoogLeNet architecture . . . 26
3.4 Description of VGGNet architecture . . . 30
3.5 Description of ResNext architecture . . . 35

4.1 A simple Confusion Matrix . . . 58
4.2 A three-class Confusion Matrix: A represents Alert, D represents Drowsy and Y represents Yawning. The first letter in each cell corresponds to the prediction, and the second letter corresponds to the ground truth, for example, DD represents samples predicted as Drowsy and are actually Drowsy. . . . 58

5.1 Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using maximum pooling. LR denotes Learning Rate, BN denotes Batch Normalization. . . . 63
5.2 Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using maximum pooling. LR denotes Learning Rate, BN denotes Batch Normalization. . . . 63
5.3 Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using average pooling and batch normalization. LR denotes Learning Rate, BN denotes Batch Normalization. . . . 65
5.4 Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using average pooling and batch normalization. LR denotes Learning Rate, BN denotes Batch Normalization. . . . 67
5.5 Results showing AlexNet model performance at learning rate of 0.001 using average pooling and batch normalization. LR denotes Learning Rate and BN denotes Batch Normalization. . . . 69
5.6 Results showing model performance of VGG-16 at learning rate of 0.001 using average pooling and batch normalization. LR denotes Learning Rate and BN denotes Batch Normalization. . . . 72
5.7 Results showing model performance of Inception ResNet V2 at learning rate of 0.001. . . . 73
5.8 Results showing model performance of ResNext at learning rate of 0.001. . . . 75
5.9 Comparing results of five trained CNN variants on 5 test videos. . . . 79

Chapter 1

Introduction

1.1 Motivation

Driving is a common activity among humans and this activity if not properly managed with its rules being adhered to can bring about dramatic consequences. Although, a lot of people participate in this activity, driving is not as simple as it might seem. It is rather a complex and potentially risky activity which requires that participants be fully concentrated psychologically and cognitively. Losing any of the psychological or cognitive concentration can result in unforeseen or unexpected consequences which include accidents (fatal or not fatal). There are several risks associated with driving among which is "road accident". Several factors could contribute to accidents on the road and some of these factors include but are not limited to: 1.) Excessive speeding; 2.) Poor operating practices including not using seat belts; 3.) Alcohol or drug use while driving; 4.) Distractions (for example talking, texting, drinking or eating); 5.) Fatigue or drowsiness; and 6.) Rash driving.

World Health Organization (WHO) statistics show that the number of death due to road traffic injuries yearly was approximately 1.25 million [1] which by 2016 had swelled up to 1.35 million yearly. This report can be interpreted to mean that approximately one person experiences (or will experience) a fatal crash every twenty four seconds [3]. This report from [3] also shows road accidents to be the leading killer of children and young ones between the ages of 5 and 29. The World Health Organization also provided in their 2009 reports that 9,587,781 vehicles were registered in South Africa with 13,768 road traffic fatalities of which 76% were male and 24% females [13]. The American National Highway Traffic Safety Administration (NHTSA) statistics also show that driver non-alertness (which includes drowsiness or sleeping and distraction such as use of phones, talking or looking elsewhere apart from the wheel) is a leading cause of motor vehicular accident. NHTSA in their 2017 reports stated that 795 lives were lost due to driver drowsiness [14] and 3450 lives were claimed in 2016 due to distracted driving [15]. A non-alert driver behind the wheel puts the life of many others on the road at risk (including other drivers and pedestrians). NHTSA's reports show that between 2011 and 2015, there were 4,121 fatal crashes involving drowsy driving with an estimated value of 90,000 motor vehicle crashes in 2015 [14]. It is however unfortunate that the rate of accidents related to drowsy and distracted drivers remain consistent over the years. Several factors could contribute to a person being drowsy: it could be deprivation of sleep, being bored or being tired as a result of a long drive or events of the day.

Drowsiness refers to a state when someone is feeling sleepy, that is, the person has a tendency to fall asleep. Drowsiness is a state between alertness and sleep and a major contributor to drowsiness is fatigue which can be caused as a result of long driving time or boredom on monotonous roads. Drowsiness level is high in the

afternoon after having lunch and higher in the midnight [16]. Consumption of alcohol or certain drugs and deprivation of sleep or sleep disorders are also other factors which can contribute highly to drowsiness.

Due to the high contribution of drowsiness to many fatal accidents, there is thus a need for a system which monitors drivers in real time and alert them when they become drowsy. According to a report by [17], the amount of vehicular accidents would be reduced by 10% to 20% if drivers and driving behaviours can be monitored in real time and given warnings based on the analysis of driver behaviour. Developing an efficient driver alertness monitoring system has both been an academic and industrial challenge. In the automobile industry, a Driver Alert Control (DAC) [18] was developed by Volvo by mounting a camera to the vehicle in order to detect lane departure thus using this metrics to evaluate driver drowsiness. Mercedes Benz [19] also has made progress in the area of drowsiness detection by monitoring drowsiness through the data obtained from the driver and his driving behaviour. Data such as steering movement correlation with driving circumstance is analyzed to check or ascertain if the driver is alert or not. BMW Active Driving Assistant [20] also analyses driver behaviour to measure drowsiness and advises the driver to take a rest through a control display showing graphic symbols. There are many other existing systems which intend to monitor drowsiness such as Audi Rest recommendation system [21], Ford Driver Alert [22], Honda Driver Attention Monitor [23], Volkswagen Fatigue Detection System [24], Bosch Drowsiness Detection System [25], Mazda Driver Attention Alert [26]. There have also been several academic contributions to developing an efficient driver drowsiness detection system. Over the years, there have been different approaches to solving the problem of driver drowsiness and these approaches can be broadly categorized into three which include: 1.) Psycho-physiological behavior of the drivers; 2.) Mode of driving of the driver; 3.) Facial information analysis based on computer vision techniques.

These categories are discussed elaborately as follows:

1. **Psycho-physiological behavior of the drivers:** includes the techniques which are based on monitoring the psychological and physiological attributes of a driver. These techniques involve using biological electronic signals such as Electroencephalography (which is a technique used to record electrical activities of the brain) [27] [28] [29] [30], Electrocardiography (which is a technique used to record electrical activities of the heart) [31] and Electrooculogram (which is a technique used to measure the cornea-retina standing potential which exists between the front and back of the eyes) [32]. This approach of using psychological and physiological attributes to detect drowsiness has proven to achieve very good performance in terms of its accuracy at identifying drowsy drivers, however, this approach has severe limitations in that this approach is not really feasible for practical application. Drivers would find it uncomfortable to wear devices with sensors all attached to their skins to measure signals just because they would be alerted when drowsy.

2. **Mode of driving of the driver:** includes techniques which uses drivers' behaviour to predict drowsiness. Studies have been conducted on observing and measuring the driving behaviour and patterns to predict drowsiness. A driver is predicted to be drowsy if he keeps driving haphazardly or in a way that is not expected of drivers to drive. Driving patterns involve, how the driver applies acceleration, how the driver stays on the lane, or braking time series. [33] [34] [35] [36] amongst many other literature are examples of research which adopt this approach of measuring drowsiness with driving pattern. This technique has proven not to be very efficient especially because the approach is not robust and not applicable in certain driving conditions [37].

3. **Facial information analysis:** This approach is a non-intrusive technique of measuring and predicting drowsiness. In this approach, an analysis is made on the facial information of the driver. Facial image analysis

can give substantial information to measure drowsiness. Information such as eye-blink frequency, movement of the head, position of the head, frequency of yawning, eye gaze movement and facial expressions can be used to distinguish between people who are in a state of drowsiness and people who are alert. [38] [39] [40] [41] [42] [43] [44] [45] [46] [16] are previous works done using facial information to predict drowsiness. Although, this approach is not the most excellent (in terms of accuracy) among the three categories, yet there is a need for a balance between comfortability, feasibility and accuracy, and the approach of analyzing visual information to predict drowsiness achieves this right balance. Some other works done using the approach of visual information include [47] [48] [37] [49] [50] [51].

The intrusiveness of the psycho-physiological approach makes it unsuitable to be used as this would be uncomfortable for drivers. The inefficiency of the driving mode and limitations to only certain driving scenarios makes this approach (driving mode) also unsuitable for a robust driver alertness monitoring system. Based on these limitations of these two approaches (psycho-physiological and driving mode), we are thus compelled to develop a non-intrusive technology which is at the same time accurate enough to monitor driver alertness in real time. Thus, in this research, we explored the approach of using visual information to monitor driver alertness in real time.

Monitoring driver alertness in real time using ordinary frame by frame classification of the visual information has been the approach of many researchers, and this approach we reckon is not the best, this is because, sometimes, a closed eye does not necessarily imply drowsiness, neither does an open mouth always imply yawning. As a result, it is pertinent to approach driver alertness monitoring by taking the facial information over a period of time into consideration.

1.2 Thesis

Driver alertness monitoring can be improved in terms of accuracy, speed and intrusion by using deep learning architectures (specifically convolutional neural networks) to predict face states (Normal, Yawning and Drowsy) in images and using specific factors (such as yawning period, eye blink frequency and eye closure duration) to form an inference on a driver alertness level based on a temporal information of the predictions from the convolutional neural networks.

1.3 Aim

The aim of this research is to use deep learning architectures (deep convolutional neural networks) to use deep neural networks to monitor driver alertness in real time in a non-intrusive manner by using visual information. In this research, we aim to use deep convolutional neural networks (LeNet-5, AlexNet, Inception-ResNet, VGGNet and ResNext-50) for face-state classification (alert, yawning and drowsy), and use specific factors (which are discussed in detail in our methodology) to penalize the alertness of a driver (which is always initially set to a value of 100).

1.4 Objectives

The objectives of this research can be summarized as follows:

1. Compare available open source face and facial feature detection algorithms and select the best suited face detection algorithm for our study.
2. Develop a face-state detection model using different Convolutional Neural Networks variants (LeNet-5, AlexNet, Inception-ResNet V2, VGGNet and ResNext-50) to train faces extracted from publicly available dataset using the algorithms selected in objective 1.
3. Develop an evaluation system which penalizes the alertness value based on predictions from the networks trained in objective 2 and return overall decision at different points in the video, for example, point at which a driver is "slightly drowsy" or "very drowsy".
4. Evaluate, compare and select best performing model based on performance metrics such as accuracy and speed of frame processing.

1.5 Research Questions

1. Which face and facial feature detection algorithms are publicly available and which is best suited for our study?
2. How can we use machine learning to detect face-states (Normal, Yawning and Drowsy) given faces as input?
3. How can we use face-states to infer drowsiness or alertness level in drivers?
4. Which is the best suited deep learning model for monitoring alertness level in drivers (in terms of speed of training and prediction and accuracy)?

1.6 Research Limitations

Our study focuses on using convolutional neural networks to monitor driver alertness. To effectively monitor driver alertness, we need to take into consideration the behavior over time, for example, yawning frequency. There are several machine learning algorithms which are best suited for monitoring sequential actions, for example, recurrent neural networks [52], however, due to limited or no access to large repository of sequence dataset, we have chosen to use convolutional neural networks. Driver alertness or drowsiness monitoring has been studied vastly in the literature, and we discuss this in detail in Chapter 2, as a result, there is sufficient publicly available data which can be used for our study.

1.7 Dissertation Structure

The remaining part of this document is structured as follows. Chapter 2 provides the related work on drowsiness detection models in existing literature. Deep Convolutional Neural Networks are discussed more elaborately in Chapter 3 and the methodology of this research is introduced in Chapter 4. Chapter 5 presents the results obtained from the methodology described in Chapter 4 and lastly, Chapter 6 presents the challenges faced during this study, recommendations for further studies and the conclusion of this study.

Chapter 2

Background and Related Work

2.1 Introduction

Driver alertness monitoring has been an academic and industrial problem over the years. In Chapter 1, we discussed how numerous researchers have proposed several methods to detect drowsiness in real time; these methods were broadly classified into three which are: **Psycho-physiological attributes** (which involves the use of sensors to measure heart rate, brain activity and cornea-retina activity), **Driving mode or driver behavior** (which includes braking time series, acceleration, lane drift etc.) and **visual information analysis** (which includes using facial information such as head pose, facial expressions, eye gaze etc). We also discussed how the industry has been trying hard to solve the problem of drowsiness; Volvo [18] and Mercedes Benz [19] have applied methods in the three categories to solve the problem of driving alertness monitoring in real time. Given the intrusive nature of the psycho-physiological attributes and not so efficient and ineffective nature of driving behavior to measure or predict drowsiness, we found that using visual information to monitor alertness is the best existing approach because of its non-intrusive nature and accuracy. Visual information such as yawning, eye gaze, head pose, facial expression etc.) are obtained from an individual and these information are fed into models which train on enough samples of data and then develop predictive abilities based on what such model trained on. Models such as this are referred to as "Machine Learning Models".

The concept of machine learning is further described in Section 2.2. In Section 2.3, we discuss a type of machine learning model - "the traditional machine learning models" and their applications to monitoring alertness in existing literature over the years. We also discussed the other type of machine learning model - "Deep Learning models" and how they have been applied to alertness monitoring in previous literature in Section 2.4. In Section 2.5, we compare the performances of the traditional machine learning algorithms and the deep learning algorithms in general and also in relation to the field of alertness monitoring. Section 2.6 concludes the chapter.

2.2 Machine Learning

Machine learning is the study of algorithms which are able to knowledge from a historical data. Machine learning algorithms learn patterns and behaviors of data and are able to make predictions or decision without requiring much of human assistance. In 1959, Arthur Samuel in his work [53] coined the term "Machine Learning". Tom M Mitchell gave a widely quoted and more formal definition of machine learning: "A computer

program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [54]. Models which are based on learning from data known as "training data" in order to make predictions without being explicitly programmed to make such particular decision in the event of unseen instance are called "Machine Learning Models". Machine learning models are broadly categorized into Supervised Learning [55] and Unsupervised Learning [56]. However, there are other machine learning methods like Semi-supervised Machine Learning [57] and Reinforcement Learning [58]. For this research, our discussion focuses on the supervised machine learning method, an approach used in our study for driver alert monitoring. This study presents a classical case where this approach would be appropriate.

2.3 Traditional Machine Learning

Over the years, many machine learning algorithms have been developed. A machine learning algorithm is one which learns or gains knowledge by training on data. These models are mathematically modeled in such a way that the algorithm improves in performance given more data. However, the traditional machine learning algorithms over time became almost constant even with more data. Traditional machine learning algorithms refer to the not-so complex machine learning algorithms which include linear classifiers, quadratic classifiers and support vectors. They are called traditional learning algorithms just to differentiate them from the quite complex ones which are referred to as Neural Networks. Traditional Machine Learning algorithms often require feature extraction before being fed into the model itself.

Examples of traditional machine learning algorithms include but are not limited to Linear Classifiers (e.g. Logistic Regression [59], Naive Bayes [60], Perceptrons) [61]; Kernel Estimation (e.g. K Nearest Neighbours (KNN) [62]); Decision Trees (e.g. Random Forests) [63]; Support Vector Machines (Least Squares SVM) [64]; Quadratic Classifiers [65]; Boosting Algorithms (e.g. AdaBoost [66], Gradient Boost [67]); Linear Regression Algorithm [68] Polynomial Regression Algorithm [69] Support Vector Regression (SVR) [70] and Regression Trees (Random Forests) [71].

2.3.1 Related Works

Traditional machine learning algorithms have seen a huge success and performance rate when applied to many real life problems such as fraud detection [72], image recognition [73], image segmentation [74] and several other computer vision tasks. Traditional machine learning algorithms have been notable for their simplicity and easiness to deploy on embedded devices. The field of alertness monitoring or drowsiness detection has also seen a large contribution of traditional machine learning algorithms to its success and deployment on many devices today. Several research has been done in which different traditional machine learning algorithms have been used. [42] is one of such research where traditional machine learning algorithm was successfully used to monitor driver alertness. [42] approached the problem of detecting drowsiness from the perspective of using visual information. The Viola and Jones algorithm [75] was used for the facial feature detection and the output of the algorithm include: "The Face, the Eyes and the Mouth". [42] trained the Viola and Jones algorithm on a thousand (1000) samples of positive images and five thousand (5000) samples of negative images. The

AdaBoost was also used to select features and train the dataset so that a well performing classifier function is achieved. Drowsiness in this research [42] was basically measured by checking the openness or closure of the eye. Experimental results showed a performance of approximately 125 ms, 15 ms and 18 ms for Face detection, Mouth detection and Eye detection respectively. The overall algorithm time was 615 ms, thus the drowsy detection model is estimated at about 450 ms time performance.

[46] and [16] also used the Viola and Jones [75] algorithm for face detection. In [16], the AdaBoost learning classifier was used to train the drowsiness detection system. Continuously adaptive mean shift algorithm was used for continuous face tracking of the driver. Experimental results in [16] produces an accuracy of about 99% which is comparatively better than the existing models at the time the implementation [16] was made. [46] on the other hand achieved an accuracy of 94%. [46] approached the problem of alertness monitoring by monitoring eye blinks. Once the face has been detected by the Viola and Jones face detector [75], the eyes region were cropped and the two eye corners and one point on the lower eye lid is computed. [46] then computed the midpoint of the two upper eye corners and the distance from the lower eye lid point. Drowsiness and alertness in this work were predicted based on the value of the distance obtained from the eye corners and lower eye lid point. If the distance is zero or near zero, the eye is interpreted as being closed and if the eye remains closed for two or more seconds, the participant is said to be drowsy.

2.4 Deep Learning

Deep Learning is a sub-category of machine learning [76]. Deep learning was inspired by Artificial Neural Networks (ANNs). ANNs were inspired by human biological neural networks. Deep learning is an approach of machine learning that mimics the human brain which works by a series of interconnected neurons. This approach helps deep learning algorithm perform better than their machine learning algorithm counterparts (that is, classical or traditional machine learning). Deep learning achieves high performance and flexibility by learning to represent the world as a nested hierarchy of concepts where each concept is defined in relation to more simple concepts and more abstract representations computed in terms of less abstract one. Contrary to traditional machine learning algorithms, deep learning requires high-end machines. Unlike traditional machine learning where a domain expert needs to identify the key features so as to reduce the data complexity and enable the algorithms to work by providing more visible patterns, deep learning learns features from data samples by itself in an incremental method, thus, removing the requirement of extraction of features by domain experts. The term "Deep Learning" was introduced by Rina Detcher in 1986 to the machine learning community [77].

Deep Neural Networks are referred to as "universal approximators" because with just one hidden layer of finite size, they can approximate continuous functions [78] [79] [80].

Deep Learning algorithms which are commonly used include: Convolutional Neural Networks (CNNs) [5]; Ordinary or Basic Neural Network(ANNs) [81]; Recurrent Neural Networks (RNNs) [52]; Deep Belief Network (DBNs) [82]; Gaussian Mixture Models - Hidden Markov Models (GMM - HMM) [83]; Recursive Neural Networks(RNNs) [84]; 'Unsupervised Neural Networks (USNs) [85]; Generative Neural Networks (GNNs) [86].

Deep learning (DL) architectures (e.g. Deep Belief Networks, Recurrent Neural Networks and Deep Neural Networks) have been successfully applied to areas of speech recognition [87], Bio-informatics, Medical Image Analysis, Computer Vision as well as Natural Language Processing. DL architectures have been notably applied to unsupervised and supervised machine learning problems just like the conventional (traditional) machine learning algorithms.

2.4.1 Related Works

Deep learning architectures have been applied to several real life problems such as Image Classification [5], Object Detection [88], Emotion Recognition [89], Scene Segmentation [84] and more specifically, to Driver Alertness Monitoring or Drowsiness Detection. [32] In their work made use of artificial neural networks (ANNs) to detect and predict driver drowsiness. [32] made research into how possible and reliable it is to use the standard sources of information which have been used to detect driver drowsiness to also predict when a given level of drowsiness will be attained. Towards improving the accuracy of the prediction and detection of driver drowsiness, [32] added more features such as driving time and the information of participants in order to see how these information can affect or influence efficient prediction and detection. In developing the model, [32] gathered information from 21 participants who were asked to drive a car simulator for one hour and 50 minutes (1 hr 50 mins) under certain conditions which were expected to induce drowsiness. Several data attributes were gathered to train the model such as psychological and behavioral attributes like frequency of blinks, heart rate and variability, rate of respiration, movement of the head and eyelids. [32] also obtained features relating to driving behaviours such as lane position, angle of the steering wheel, speed at points in time and time to lane crossing. [32] developed two artificial neural network models - one for detecting the degree of drowsiness each minute and the other for predicting every minute the estimated time at which a the driver would reach a certain level of drowsiness level. This study by [32] showed a success in detecting drowsiness level with a mean square error of 0.22 and mean square error rate of 4.18 minutes when predicting that a given level of drowsiness (moderately drowsy) would be reached. [41] also proposed a method of using deep neural networks to detect real time driver drowsiness. A major research conducted by [41] is how to implement a drowsiness detection model on an android device. Given the large size of deep neural network models, [41] proposed a method of compressing of a heavy baseline model to a light weight model. [41] made use of the dataset provided by the National Tsing Hua University (NTHU) [90] which comprised of 22 participants of different ethnicity. The information was obtained from these participants under several conditions such as day and night illuminations and training features such as driving mode, slow blink rate, conscious laughter, yawning and dozing were extracted. The multi-layer perceptron was trained on 18 participants while the model was evaluated on 4 participants. There were majorly five driving scenarios which include participant with glasses in the day, participants without glasses in the day, participants with sunglasses in the day, participants with glasses in the night and participants without glasses in the night. Experimental results shows that the model achieved a good performance with over 80% accuracy rate after compression.

[30] approached the problem of detecting drowsiness in drivers from the perspective of using psychological attributes. Visually Evoked Potentials (VEP) were obtained and computed from electroencephalographic (EEG) signals from the visual cortex. [30] however used these features to build the drowsiness detection model using classical artificial neural networks. Results from this study showed a performance of 97% in terms of accuracy on the test data.

In [39], drowsiness detection model was built using the classical artificial neural network which was trained on visual information. Although, the Viola and Jones [75] method was used for face detection after which the eyes are then extracted from the face which was detected in the image. The eyes are then fed into the ANN to classify them into Closed or Open. Closed eyes are considered to mean the driver or participant is drowsy while open eyes mean otherwise. Experimental results showed that building a drowsiness detection model with ANNs trained on visual information such as the eyes extracted from the face using Viola and Jones method can be quite effective up to 93% accuracy.

[37] is one of the notable studies which have plainly used deep learning to build an efficient driver alertness monitoring system. [37] proposed using Convolutional Neural Networks (CNNs) firstly for face detection after which the output (face detected) is fed into another Convolutional Neural Network for drowsiness detection. The Multi Task Cascaded Convolutional Neural Network (MTCNN) [91] was used for face detection. Also, [37] much more focused on deploying this proposed model into embedded devices and as such proposed using a method of compressing heavy baseline models to light weight models deployable to an embedded board. [37] obtained visual information such as the eyes and the mouth by extracting a patch of image around the facial landmark obtained from MTCNN in order to make predictions whether the driver is drowsy or not. After compressing the heavy baseline model to work well on embedded devices, the model still achieved a good performance measured at 89.5% accuracy of a 3-class classification (Drowsy, Yawning and Not Drowsy) with a speed of 14.9 Frames per second (FPS) on Jetson TK1.

2.5 Comparison of Deep Machine Learning and Traditional Machine learning

Traditional machine learning algorithms (such as the Support Vector Machines and the AdaBoost) have been successfully applied to the problem of alertness monitoring. The Haar cascade classifier (Viola and Jones algorithm) [75] has proven to be a very good and fast face detector and the AdaBoost and Support Vector Machines (SVM) have also achieved good performance in training a driver alertness monitoring system (or drowsiness detection system). One of the major advantages of using the traditional machine algorithms over the deep learning algorithms observed from our literature survey is that the traditional machine learning classifiers are light weight models thus allowing for easy deployment on embedded devices whereas deep learning classifiers are heavy weight and can be quite difficult to deploy such models on embedded devices. The deep learning classifiers have however been notably known for their high performance and recognition rate and have shown their superiority over traditional machine learning classifiers especially in the fields of Computer Vision, Natural Language Processing (NLP), Speech Recognition, and Gaming. The question however is: "What good is a perfect system that proves difficult to deploy"? There have been several attempts to solve this problem because despite the commendable high performance rate, one would need high computation resources to deploy the deep learning models. Several approaches towards reducing the heavy weight deep learning models for easy deployment have been proposed in [92] [93] [94] [95]. One of such approaches is to apply a procedure called "Pruning" [92]. Many trained networks often consist of redundant information. Some of these redundant information can be removed thus reducing the model size. This process of removing redundant information is called "Pruning". Quantization methods have also been introduced to reduce model size, Bit Quantization is one of such methods. In Bit Quantization, the least number of bits are used to represent the model information while minimizing loss of accuracy. R. Caruana et al. [96] also proposed the method of using ensemble selection from the libraries of models to reduce the model size. Another recent proposed approach is the Knowledge

Distillation approach between two networks. [37] [97] [98] [99] [100] used this approach in reducing model size and results show a yet very good performance in speed, detection rate and size.

2.6 Conclusion

This Chapter sought to establish the context of this research. The framework for driver alertness monitoring was established in Section 2.1 where we hinted about the commendable approach of a non-intrusive technology to monitor driver alertness. In Section 2.2, we discussed the concept of machine learning and elaborated on the broad two categories of machine learning (Classical or traditional machine learning and Deep learning) in Section 2.3 and Section 2.4. We further investigated how driver alertness monitoring has been approached in previous literature and observed that the systems developed to monitor alertness are not very efficient, especially because of the methodology used. We also compared the performance of Deep learning against Traditional machine learning in previous literature and how there have been attempts to improve the deployment of large deep learning models to embedded devices in Section 2.5. The outstanding performance of deep learning in terms of its accuracy, generalizability and speed was explained in Section 2.5 and we observed from literature that though these deep learning models can be quite complex and resources consuming thus becoming heavy weight, there have been successful model size reduction techniques. With this discovery, we decided to implement a driver alertness monitoring through the use of deep learning architectures thereby improving the performance of driver alertness monitoring systems. Our findings in this chapter will provide the context for the methodology of the research in the next chapter.

Chapter 3

Deep Convolutional Neural Networks

3.1 Introduction

Convolutional Neural Networks (CNNs) are a type of feed forward Artificial Neural Networks (ANN). Artificial neural networks in machine learning were inspired by biological neural networks which constitute animal brains [101]. The artificial neural network is a collection of units or nodes which are connected to one another, these connected units or nodes are called "Neurons", and like in the biological brain, information (or signals) can be sent from one neuron to another neuron connected to it.

Convolutional Neural Networks were officially introduced by Yann Lecun et al., (1998) in their paper [102]. Prior to this, there were convolutional neural networks though by a different name [101]. Unlike traditional artificial neural networks which take in vectors as input, convolutional neural networks are able to exploit the nature of other types of data, for example, 1-Dimensional Data such as text, 2-Dimensional Data, such as Images, or 3-Dimensional Data such as MRI scans. CNNs nature allows for encoding image-specific features thus making the network more suited for image-related tasks while reducing the parameters required to set up the model. CNNs like every feed-forward neural network send data from the input layer through to the output layer. Initially, a map of virtual neurons is created with random weights assigned to each neuron or connections between neurons. The input is multiplied by the weight, returning a value between 0 and 1, the CNN keeps doing this to adjust the weights until there is an optimal correct mathematical manipulation which models the data correctly.

Deep Neural Networks (DNNs) are more sophisticated neural networks which are able to learn functions that are very complex. The DNNs are called "Deep" because of the several hidden layers between the Input and Output layers. DNNs can learn complex non-linear relationships. There are different DNNs which have been introduced over time in literature and these DNN architectures differ from one another based on how the hidden layers work. One of the most common DNNs is the Deep Convolutional Neural Networks (DCNN). Deep CNNs (or DCNNs) are similar to CNNs except that, in Deep CNNs, there are more hidden layers present, however, Deep CNNs and CNNs both have in common at least, the Convolutional Layer, Pooling Layer, and Fully Connected Layer. CNNs perform well on image data because they employ a method called "Convolution" to extract features unlike in traditional neural networks (Multi Layer Perceptrons) which use separate feature extraction techniques such as Local Binary Patterns (LBPs), Surf, Histogram of Oriented Gradients (HOGs) etc.

Convolution is a commonly used mathematical concept in digital signal processing when performing operations on signals which take a time series form. Basically, when performing convolution, we simply attempt to blend

two functions in a coherent manner. Convolution can be described mathematically for a discrete domain of one variable as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a), \tag{3.1}$$

where w represents the kernel, s is the feature map and x represents the input data.

Equation 3.1 describes a simple convolution which applies a 1D kernel on a 1D signal, however, in computer vision applications, such as the driver alertness monitoring, we make use of 2D input signals. Convolution on a 2D image I using a 2D kernel K can thus be mathematically represented as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.2}$$

Convolution is commutative, so we can as well represent equation 3.2 as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{3.3}$$

Although, convolutional neural networks are so called, the operations performed on image inputs are not exactly convolution, rather a modified variant which is called cross correlation where the input is time reversed but the kernel is not. Cross correlation can be mathematically expressed as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{3.4}$$

Convolutional Neural Networks were developed to improve on the limitation of the traditional neural networks as the traditional neural networks struggle with the computational complexity necessary to compute image data. Traditional neural networks are very suitable for handling tasks with quite low dimensional data but when the dimension of an input increases, the weight on just a single neuron increases substantially. In CNNs, the neurons are organized into three dimensions which include the depth, width and height.

Overall Architecture of CNNs Convolutional Neural Networks basically have three types of layers: **Convolution Layers**, **Pooling Layers** and **Fully Connected Layers**.

These layers (convolution, pooling and fully connected) are combined together to form the convolutional neural network. A simple CNN architecture for an image classification is illustrated in Figure 3.1.

A Deep CNN (DCNN) consists of more layers in between the input and output layers. Below, we discuss the layers and parameters used in DCNNs:

- **Input Layer:** This layer holds the pixel values of the image to be processed. The input here is expected to be a 3D (that is, RGB image).

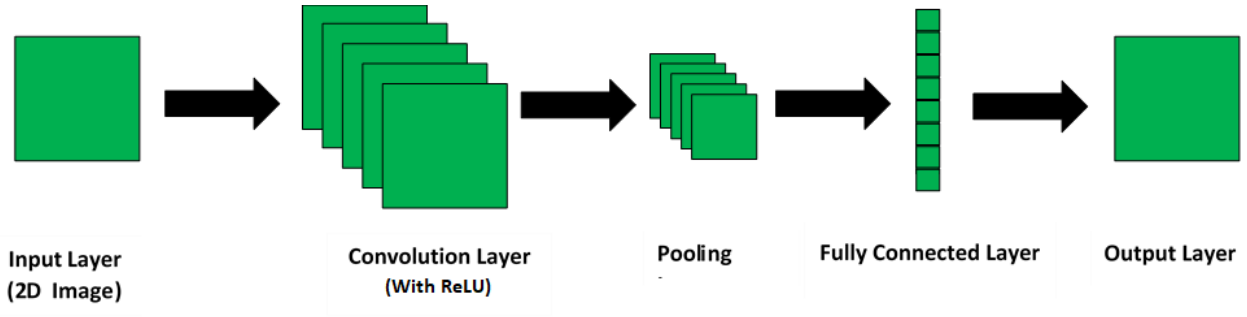


Figure 3.1: A simple convolutional neural network architecture consisting of five layers which are: 1.) Input Layer, 2.) Convolution Layer, 3.) Pooling Layer, 4.) Fully Connected Layer and 5.) Output Layer.

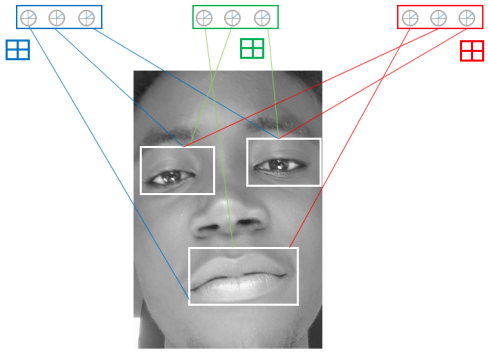


Figure 3.2: Parameter sharing in convolution layers

- **Convolution Layer:** The core part of CNNs is the convolution layer. The output of neurons connected to the local regions is computed in this layer by getting the dot product of the weights and a small region where the neurons are connected to in the image. If a number of n $K(i,j)$ kernels are used in the convolution layer over an image $I(i,j)$, the resulting volume is then $I(i,j,n)$, where n is the depth of the resulting convolution. The layer parameters in the convolution layer focuses on using learnable kernels. The kernels are basically small in spatial dimensionality (height and width) but follows with depth of the input. When the input data is passed through the convolution layer, a kernel (or filter) is used to convolve around the spatial dimensionality of the input data and produces a 2D activation map. The kernels used here all have corresponding activation maps which are combined along the dimension of the depth thereby producing an output volume. In convolutional layers, neurons are not entirely all connected to one another, rather, a neuron only has connections to a specific part in the image. This is to ensure performance in training and allow for easy computation thus reducing computation complexities. The convolutional layer has two significant importance: Parameter Sharing and Local Connectivity. By convolving the input image with a kernel, the kernel parameters are thus shared for all local positions of the image; thus resulting in a reduction of the network's parameters as it learns features of the image which can be used in similar positions without having to relearn those features[103]. This is called parameter sharing, and it is graphically represented in Figure 3.2. The local connectivity helps the network to learn kernels which connect to a local region of the input image thereby exploiting the spatial local correlation of the input image, because a pixel has more correlation to its surrounding pixels than to more distant pixel[103]. Figure 3.3 shows a graphical representation of local connectivity

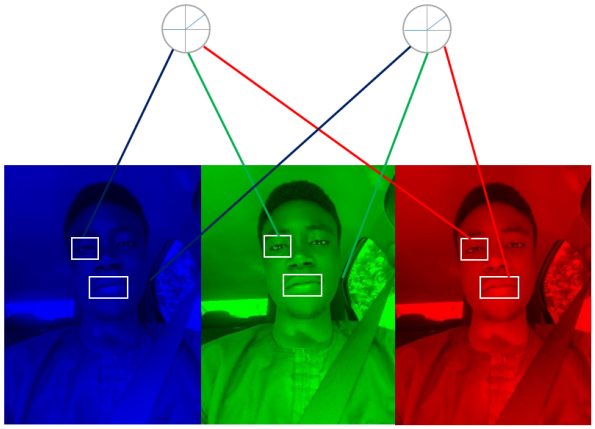


Figure 3.3: Local connectivity in convolution layers

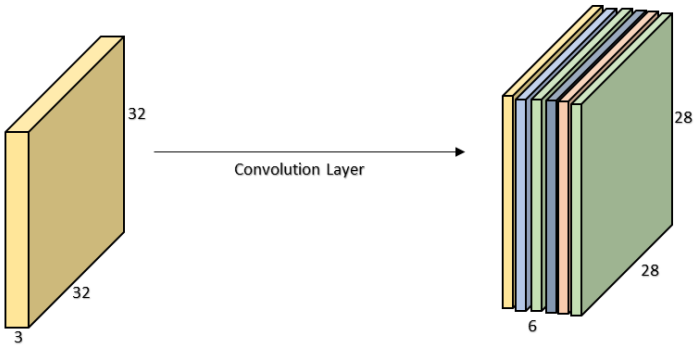


Figure 3.4: Activation map

in convolutional layers.

The output volume from the convolution of an input volume with a kernel is called an **Activation Map (or Feature Map)**. The activation map size is dependent on the size of the kernel, for example, sliding a "kernel of size 3×3" over a 32×32 image will result in a 30×30 activation map because there are only 900 locations that the 3×3 kernel can fit on the 32×32 input image. The depth of the activation map is the number of kernels used, for example, sliding a $3 \times 3 \times d$ kernel results in an activation map of $30 \times 30 \times d$ volume. Figure 3.4 shows a graphical representation of an activation map obtained from the convolution of a $32 \times 32 \times 3$ image with a 3×3 kernel. The output from the convolution layer can also be further optimized by three hyper-parameters which are the strides, activation function, zero padding, and the depth.

- ***Stride***: The stride is basically defined as how we move the filter or kernel across the spatial dimensionality of the input. Increasing the stride value helps to reduce the amount of overlapping thus producing an output with low spatial dimensionality while using a small stride means we move the kernel over the image little by little, resulting in more overlapped receptive fields thus producing a larger output volume compared to using a larger stride. For example, using a stride of 1, applying a 3×3 kernel on 32×32 image will result in a 30×30 output volume whereas a stride of 2 will result in 15×15 output volume.

- ***Depth***: The depth of a volume refers to the channels in the volume, for example. an RGB input image x_i of "size $32 \times 32 \times 3$" has a depth of 3 and if a $3 \times 3 \times 10$ kernel is applied on the input image, it results in an output volume of $30 \times 30 \times 10$, thus increasing the depth of the volume to 10.

- ***Zero Padding***: Zero padding helps to preserve the spatial dimension of an image, for example, after applying a "kernel of size 3×3" on a 32×32 image, the output being 30×30 can be zero-padded, that is, we add zeros to the borders, thus making the "size 32×32". The resulting output is computed as [104]:

$$\text{Output} = \frac{I - K + 2P}{S + 1}, \tag{3.5}$$

  where I is the input volume (height $\times$ width $\times$ depth); K is the kernel size, S is the stride value and P is the Zero padding value.

- ***Activation Function:*** Activation functions also called "transfer functions" are used to determine the output in neural networks. The activation function can be applied at the end of a neural network or in-between. In some CNN architectures, the activation functions simply takes numerical values as input and performs a non-linear transformation on the input and produce an output, this output is then fed as an input to the next layer. Different activation functions can be used, some include Tanh, Rectified Linear Unit (ReLU), and Sigmoid. However, the ReLU has proven to work better than the Sigmoid or Tanh as they increase the convergence of Stochastic Gradient Descent (SGD) by a factor of 6 [5]. The **Rectified Linear Unit (ReLU)** applies an element-wise activation function (for example, the sigmoid activation) to the result of the activation produced by the former layer.

  The ReLU can be mathematically described as:

$$ReLU(x) = \max(0, x). \tag{3.6}$$

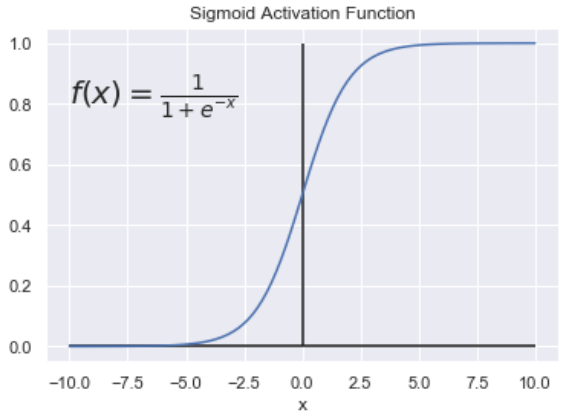
  ReLU thus converts the input function to zero if the input is negative and leaves the input function unchanged if positive. The major advantage of ReLU over other activation functions such as Tanh and Sigmoid is that, it does not activate all the neurons at the same time [105] thereby allowing for easier computation. We show a graphical representation of the Sigmoid, Tanh and ReLU activation functions in Figure 3.5.

  Figure 3.6 (1) shows a visual representation of how convolution is performed on an input image in the convolution layer. Figure 3.6 (2) shows how the kernel is moved over the input image using 1 stride and 2 strides and Figure 3.6 (3) shows how zero padding is performed on an image.

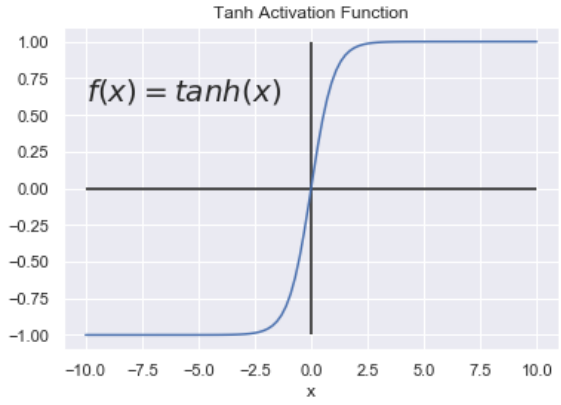
- **Pooling Layer:** This layer is used to down-sample the spatial dimensionality of the input image, thus reducing the number of parameters in that activation. There are several options in the pooling layer. The *max pooling* option is the most commonly used and it works basically by taking a patch and applying this patch to an input volume. The maximum number in all sub-regions where the kernel convolves around is selected as the output. Figure 3.7 presents a graphical representation of maximum pooling .

  There are other pooling options available for use among which are the L2-norm pooling and mean (mostly called average) pooling . The spatial dimension (height and width) are reduced only, that is, the depth of the input remains unaffected. There are majorly two reasons for this:

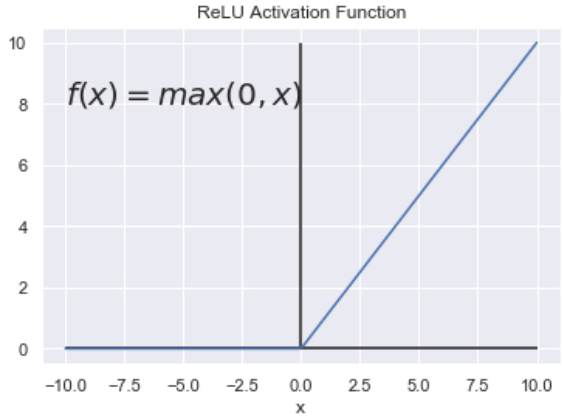
  1. A reduction by 75% of the parameters or weights occurs, thereby reducing the cost of computation.
  2. There is a control of over-fitting thus improving generalization performance.



(a) Sigmoid



(b) Tanh



(c) ReLU

Figure 3.5: Commonly used activation functions

- **Fully Connected (FC) Layer:** This layer is similar to how the traditional artificial neural networks (ANNs) work. In this layer, class scores from the activation are produced for classification. The rectified linear unit (ReLU) can also be used between the fully connected layers in order to improve the performance of the model.

Convolutional Neural Network architectures are basically formed by combining these layers together making optimal use of the hyperparameters. To improve the overall model's performance, it is recommended that the number of convolution layers be increased thus allowing more complex features of the input vector to be selected. A major advantage of the CNNs over the traditional ANNs is that CNNs help to reduce complexity issues faced with when feeding a large input. It thus would be wise to make use of this advantage by setting optimal hyperparameters to effectively make use of this advantage. Complexities can be further reduced in the convolution layer by making use of many smaller sized convolution layers. As long as the input image is not exactly too large, a good CNN architecture can make use of small kernels with a stride of 1 in the convolution layer. If the image however, is very large (considerably over 128×128), then it would be better to use large kernel sizes and a stride of 2 or 3 depending on how large the input image is [104].

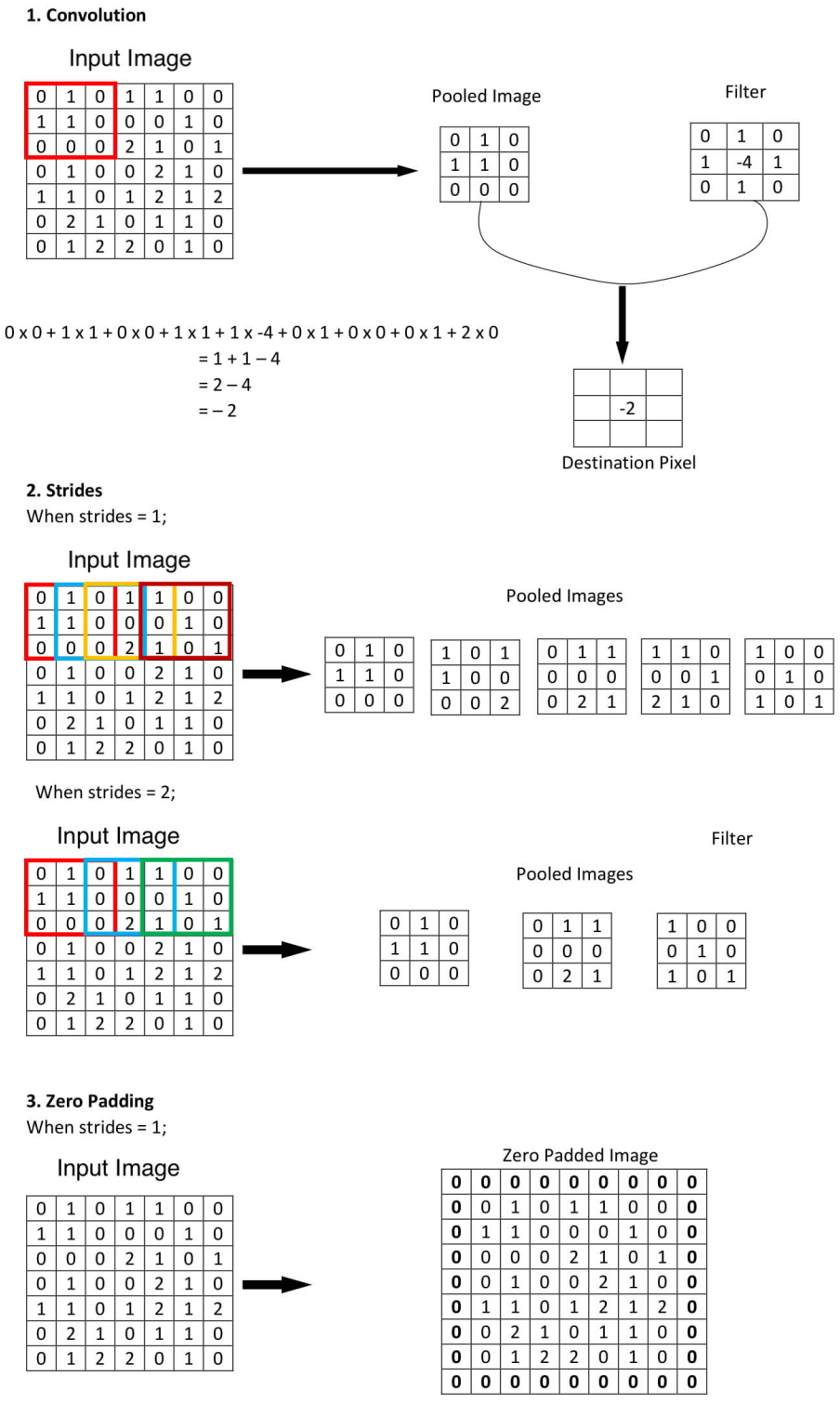


Figure 3.6: Convolution, strides and zero padding



Figure 3.7: An example of maxpooling using a "patch of size 2×2" at stride of 2.

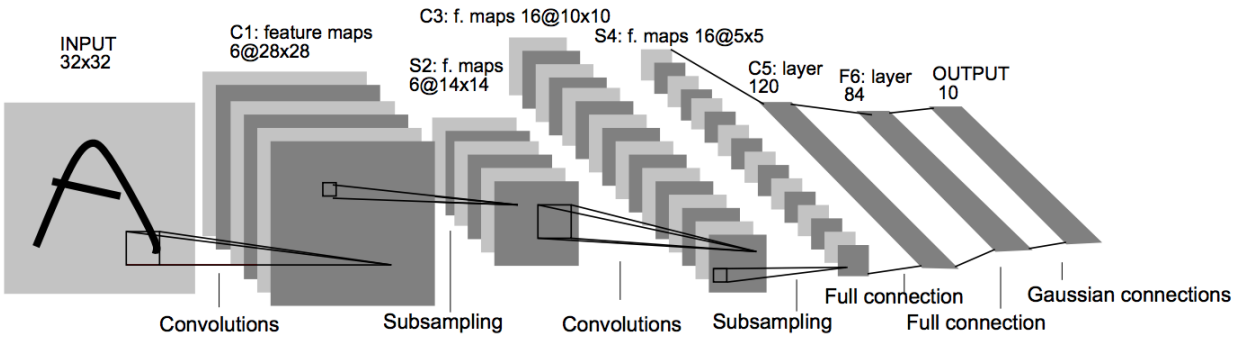


Figure 3.8: LeNet architecture by Yann Lecun et al., (1998) [4]

3.2 Convolutional Neural Network Architectures

There are many convolutional neural architectures which have been used over the years, however, for the purpose of this research, we would be discussing only five variants of convolutional neural networks. These convolutional architectures include: LeNet-5, AlexNet, Inception-ResNet V2, VGGNet and ResNet. These five architectures have been famously used in literatures because of their performances. We will thus be developing these five variants of convolutional neural networks to detect driver drowsiness.

3.2.1 LeNet-5

The LeNet-5 neural network architecture which was especially used on MNIST handwritten recognizer patterns and achieved very commendable performance. In 1998, Lecun et al.,(1998) [4] proposed the LeNet-5 which was trained on a 32×32 pixel grayscale input image. The LeNet-5 architecture was proposed at a time when there was no large computing power, thus the technique was not scalable to high dimensional images. Figure 3.8 shows a graphical representation of Yann Lecun's LeNet-5 architecture. The LeNet-5 is given an input grayscale image $I(i, j)$ of "size $32 \times 32 \times 1$". The neural network uses a kernel $K(i, j)$ with a stride $S(i, j)$. The resulting output volume (O_i,O_j is 28×28. The Output_i and Output_j are calculated using the formula below:

$$\mathrm{O}_i = \left[\frac{I_i - K_i + 2P}{S_i}\right] + 1, \tag{3.7}$$

$$\mathrm{O}_j = \left[\frac{I_j - K_j + 2P}{S_j}\right] + 1, \tag{3.8}$$

where I_i is the image width; I_j is image height; K_i and K_j are kernel width and height, S_i and S_j are Stride width and height, and P is Pooling. The pooling layer is the next layer after convolutional layer. Using a stride of 1×1, the input volume 28×28 thus becomes downsampled to 14×14. The output matrix which is now 14×14 is then fed to the FC layer which has 120 units, followed by another fully connected layer with 84 units. The output class of the dataset which [4] trained on has 10 possible values which range from 0 to 9.

LeNet-5 architecture consists of 7 layers excluding the input layer. The layers and their operations are discussed in Table 3.1.

Layers	Description
Layer 1	The input image to the convolutional layer is of "size $32 \times 32 \times 1$". 6 "kernels of size 5×5" are applied using a stride of 1. The resulting output is thus a $28 \times 28 \times 6$. **Total parameters** $= 5 * 5 * 6 + 6$**(bias)** .
Layer 2	This layer is a pooling layer and here we apply 6 "kernels of size 2×2" using a stride of 2×2. The input matrix to this layer being $28 \times 28 \times 6$ thus gets downsampled to $14 \times 14 \times 6$. **Total parameters** $= [1$**(trainable parameter)** $+ 1$**(trainable bias)**$]$.
Layer 3	Similar to the first layer, this is a convolutional layer. 16 "kernels of size 5×5" is used to convolve the input volume. With the stride value being 1, the input matrix to this layer which is $14 \times 14 \times 6$ is convoluted upon and the resulting output becomes $10 \times 10 \times 16$. **Total parameters =** $5 * 5 * 16 + 16 = 416$.
Layer 4	This layer is a pooling layer similar to the second layer . In this layer, we apply 16 "kernels of size 2×2" using a stride of 2. The input to this layer being a $10 \times 10 \times 16$ is thus downsampled to $5 \times 5 \times 16$. We then apply the activation function (sigmoid) on the downsampled volume. **Total Parameters** $= (1 + 1) * 16 = 32$.
Layer 5	Again, similar to the first and third layers, we have a convolutional layer here, only that we apply 120 "kernels of size 5×5" instead. The input to this layer is a $5 \times 5 \times 16$ thus giving an output of $1 \times 1 \times 120$. **Total parameters** $= 5 * 5 * 120 = 3000$.
Layer 6	This is a fully connected layer with 120 units, bringing total parameters to 10164. **Total parameters** $= 84 * 120 + 84 = 10164$.
Layer 7	This is the final layer and it is referred to as the output layer. This layer is a fully connected layer and it has 10 units which represent the 10 possible values 0 to 9. **Total parameters** $= 84 * 10 + 10 = 924$.

Table 3.1: Description of LeNet-5 architecture

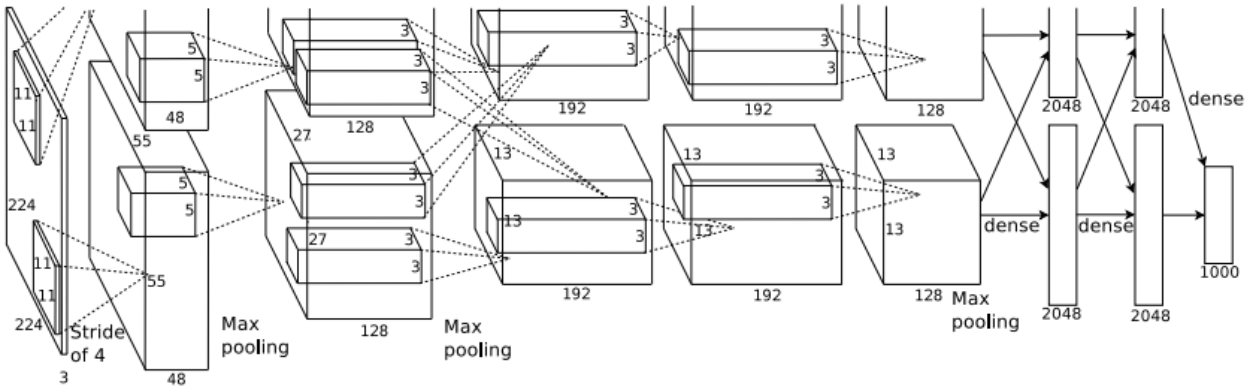


Figure 3.9: AlexNet architecture by Alex Krizhevsky et al., (2012). [5]

3.2.2 AlexNet

The AlexNet is a convolutional neural network architecture variant and is similar to the LeNet-5 architecture, except that, the AlexNet is deeper with many convolutional layers and more kernels. Alex Krizhevsky, Geoffrey Hinton and Ilya Sutskever (members of the supervision group) designed the AlexNet in 2012.

In the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), the AlexNet architecture won the challenge with an outstanding performance [106] with a reduction in the error from 26% to 15.3% of the runner-up most performing model. The second place model had an error rate of 26.2% but it was not a variant of CNN architecture. To accelerate the training speed by more than 5 times, AlexNet uses the rectified linear unit (ReLU) activation (instead of the tanh) function to include non-linearity resulting into an increase in accuracy of the model. In order to reduce over-fitting, AlexNet uses dropout regularization to prevent complex co-adaptations on training data. The AlexNet architecture consists of 5 convolutional layers and three FC layers, with ReLU activation used in the FC layers and convolutional layer. The dropout regularisation is applied before the first and second fully connected layers only. The total parameters in AlexNet is over sixty two million, thus in one forward pass, it requires over a billion computation units. Alex Krizhevsky et al., (2012) specified in their research that the AlexNet architecture trains ninety epochs for about a week on GPUs. The AlexNet architectural framework is described diagrammatically in Figure 3.9.

AlexNet architecture consists of 8 layers excluding the input layer. The layers and their operations are discussed in Table 3.2.

Layers	Description
Layer 1	The first layer is the convolutional layer. An input image of "size $227 \times 227 \times 3$" is fed into the convolutional layer. 96 "kernels of size 11×11" with a stride of 4 is applied on the input image. The resulting output volume is thus applied on the input image. The resulting output volume is thus $55 \times 55 \times 96$ which is then downsampled using maxpooling . The maximum pooling is applied using a "kernel of size 3×3" at a stride of 2. The resulting output volume thus becomes $27 \times 27 \times 96$.

Layers	Description
Layer 2	The next layer is a convolutional layer with an input matrix of $27 \times 27 \times 96$. A "kernel of size 5×5" is applied with "SAME" padding thus resulting into same size of 27×7 but with a depth of 256 (that is, output volume = $27 \times 27 \times 256$). This output is again downsampled using maxpooling. The maxpooling is applied here using a "kernel of size 3×3" at a stride of 2. The resulting output volume is thus $13 \times 13 \times 256$.
Layers 3,4,5	The third, fourth and fifth layers are a stack of convolutional layers and they all use a 3×3 kernel. The $5th$ layer makes use of "SAME" padding with a value of 1. The output volume from the third Layer is $13 \times 13 \times 384$ which is convoluted on in the fourth layer and the output volume remains $13 \times 13 \times 384$. This is immediately followed by another convolutional layer which thus gives an output of $13 \times 13 \times 256$. The output from the $5th$ layer is then downsampled using maximum pooling. A "kernel of size 3×3" at a stride of 2 is used and the final resulting output volume is $6 \times 6 \times 256$.
Layers 6,7,8	These layers are are "fully connected layers". In the sixth layer, we have 9216 neurons which are connected. The rectified linear unit activation function is applied on the output from the sixth layer and then fed to another fully connected layer which is the seventh layer. This layer comprises of 4096 neurons with the rectified linear unit again applied on the output. The final layer is the eighth layer which comprises of 1000 neurons.

Table 3.2: Description of AlexNet architecture

3.2.3 GoogLeNet/Inception

The GoogLeNet (also called Inception) was inspired by LeNet-5 introduced by Yann Lecun et al., (1998). The GoogLeNet was introduced by researchers at Google in 2014 and the network won the ImageNet challenge for classification and detection tasks in 2014. The GoogLeNet model comprises of a primary unit which is referred to as an "Inception Cell" where several convolutions are done and at varying scales and then we combine the outputs produced. To optimise computational costs, the researchers reduced the input channel depth by using 1×1 convolution. A series of 1×1, 3×3 and 5×5 kernels are used in each inception cell in order to extract significant information about an input image at different scales. In order to maintain the size of the output for easy concatenations, SAME padding is applied when at the max-pool layer. In Figure 3.10, we show an example of an inception module.

The GoogLeNet architecture consists of 22 layers in depth and reduces the number of parameters in AlexNet which was 60 million to 4 million.

In 2014, the GoogLeNet achieved a top-5 error rate of 6.67% which was very close to human level performance thus winning the challenge. This compelled the organisers of the ILSVRC to evaluate the model and after training a human expert, Andrey Karpathy, he (Andrey) managed to achieve a top-5 error rate of 5.1% on single model and 3.6% on ensemble. The GoogLeNet architecture used a convolutional neural network architecture which was inspired LeNet (developed by Yann Lecun et al., (1998)), however, the GoogLeNet researchers improved the overall model by including what was termed "inception module". This module made use of Batch normalisation, Image Distortions and Root Mean Square (RMS) propagation. Figure 3.11 shows the improvement which was made to the inception module, this module features the dimensionality reduction technique applied.

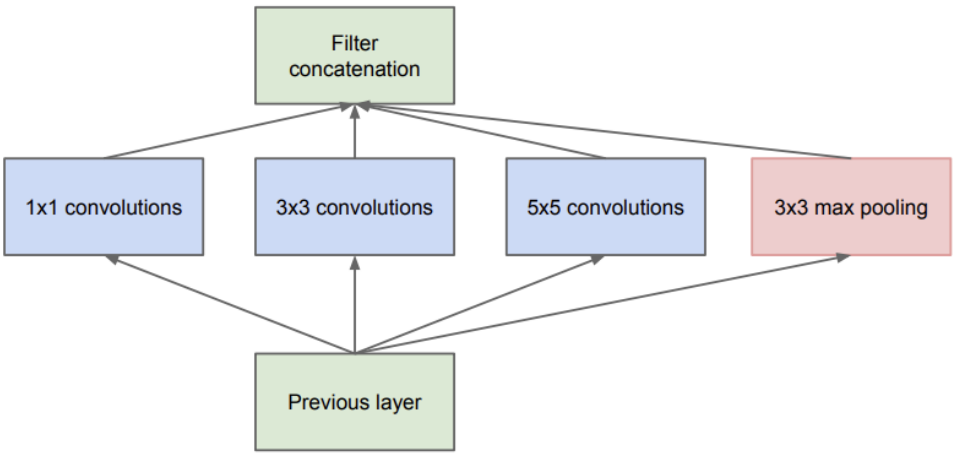


Figure 3.10: An example of a basic inception module. [6]

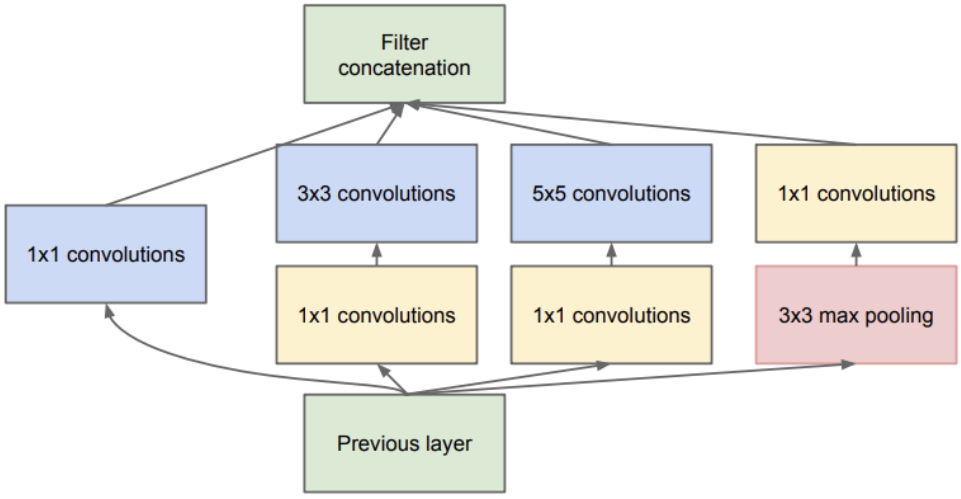


Figure 3.11: Inception module with dimensionality reduction. [6]

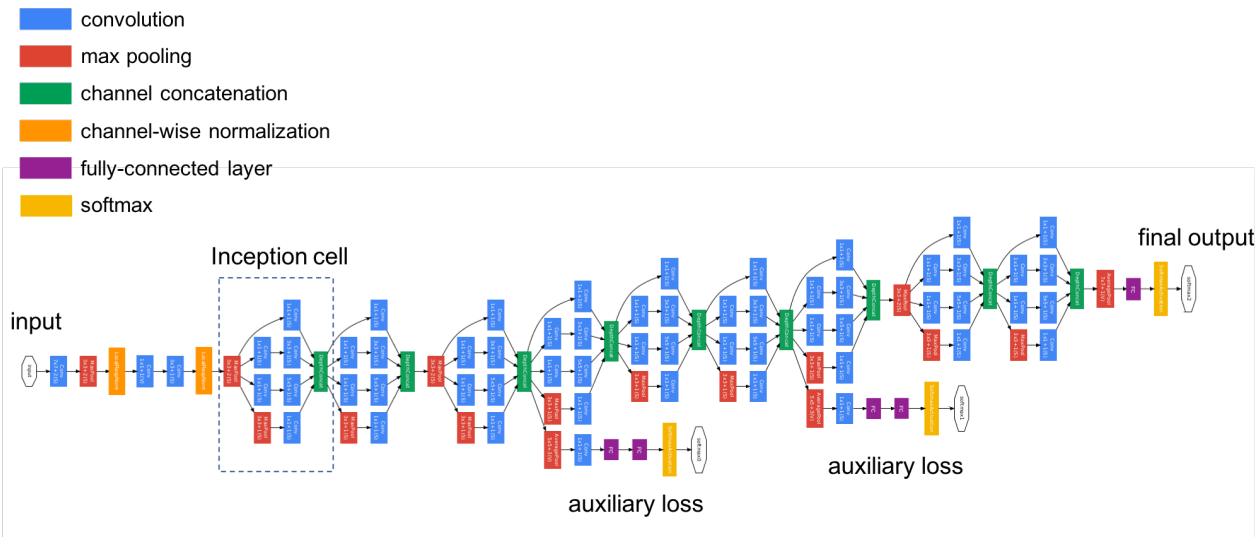


Figure 3.12: GoogLeNet (Inception) architecture used for object classification. [6]

The major aim of using this architecture was to reduce the number of parameters by using very small convolution instead. The researches proved that the use of very small kernels would be better than large kernels. While it is true that the convolution with large spatial kernels (e.g. 5×5 or 7×7) are very good at extracting features on a large scale, it is however is too expensive. To avoid this expense, it was suggested to cheaply represent a 5×5 convolution by two stacked 3×3 kernels, for example, a $5 \times 5 \times d$ kernel would require a 25d parameters while two $3 \times 3 \times d$ kernels would require only $18d$ parameters. However, for us to accurately represent the 5×5 kernel, we shouldn't use any non-linear activation between the two 3×3 layers. It was however discovered that the linear activation was always low in performance compared to using rectified linear units (ReLU) in all stages of factorisation. The researches also showed that the 3×3 convolutions can as well be cheaply represented as 3×1 and 1×3 convolutions. In summary, the researches and developers of GoogLeNet architecture proposed that we can always compute an $n \times n$ convolution as a $1 \times n$ convolution which is followed after by another $n \times 1$ convolution thus giving an efficient solution. The overall GoogLeNet architecture is shown in Figure 3.12.

GoogLeNet architecture consists of 22 layers in deep. The GoogLeNet architecture is represented in a tabular format in the Table 3.3. A revised version of the GoogLeNet which makes use of the efficient inception module is shown in Figure 3.13.

There have been several revisions of the GoogLeNet such as GoogLeNet (Inception V1), Inception V2, Inception V3, Inception V4, Inception ResNet V1 and Inception ResNet V2. The Inception ResNet V1 has a similar computational cost to Inception V3 while Inception ResNet V2 has a similar computational cost to Inception V4. The Inception ResNet V1 and V2 both have the same structure for the modules and reduction blocks.

Layers	Description
Layer 1	The first layer is the Convolutional layer. In this layer, we feed an RGB image as input and convolve with 64 "kernels of size 7×7" using a stride of 2. The output from this layer is of "size $112 \times 112 \times 64$". This output is then fed to a maximum pooling layer with a patch size of 3×3 using a stride of 2. The output from the pooling layer is of "size $56 \times 56 \times 64$".
Layers 2 and 3	An input of $56 \times 56 \times 64$ is fed from the pooling layer to the second layer which is a convolutional layer. In this layer, we use 64 "$\#3 \times 3$ reduce kernels" and "192 kernels of size 3×3". The $\#3 \times 3$ reduce kernels are 1×1 kernels in the reduction layer which are used before 3×3 convolution. This reduction layer uses rectified linear unit activation function as well as all other subsequent reduction layers. The output of layer 2 is fed to convolutional layer 3 which performs operations exactly similar to the second convolution, that is, 64 $\#3 \times 3$ reduce kernels and 192 "kernels of size 3×3". The output from the third layer is of "size $56 \times 56 \times 192$", which is then fed to another maximum pooling layer using a "patch of size 3×3" at a stride of 2. The output size thus becomes $28 \times 28 \times 192$.
Layers 4 and 5	These layers are called inception $(3a)$ layer. In inception $(3a)$ layer, we use perform two convolutions using 64 "kernels of size 1×1", 96 "kernels of size $\#3 \times 3$ reduce", 128 "kernels of size 3×3", 16 kernels of size $\#5 \times 5$ reduce and 32 kernels of size 5×5. Similar to the $\#3 \times 3$ reduce, the $\#5 \times 5$ reduce kernels are 1×1 kernels in the reduction layer which are used before 5×5 convolution. The output volume is of "size $28 \times 28 \times 256$".
Layers 6 and 7	Layers 6 and 7 are called inception $(3b)$ layer. This layer follows immediately after inception $(3a)$ layer. In inception $(3b)$ layer, we perform two convolutions using 128 "kernels of size 1×1", 128 "kernels of size $\#3 \times 3$ reduce", 192 "kernels of size 3×3", 32 kernels of size $\#5 \times 5$ reduce and 96 kernels of size 5×5. The output volume is of "size $28 \times 28 \times 480$". The output from the inception $(3b)$ layer is then pooled using a patch size of 3×3, thus giving an output volume of "size $14 \times 14 \times 480$".
Layers 8 and 9	We further feed the output from the pooling layer to layers 8 and 9 which are called inception $(4a)$. Here, we perform two convolutions similar to other inception blocks, except that we use 192 "kernels of size 1×1", 96 "kernels of size $\#3 \times 3$ reduce", 208 "kernels of size 3×3", 16 kernels of size $\#5 \times 5$ reduce and 48 kernels of size 5×5, resulting in a volume of "size $14 \times 14 \times 512$".
Layers 10 and 11	The inception $(4a)$ is immediately followed by another inception block called inception $(4b)$. The inception $(4b)$ is made up of two convolutional layers using 160 "kernels of size 1×1", 112 "kernels of size $\#3 \times 3$ reduce", 224 "kernels of size 3×3", 24 kernels of size $\#5 \times 5$ reduce and 64 kernels of size 5×5, resulting in a volume of "size $14 \times 14 \times 512$".
Layers 12 and 13	The inception $(4c)$ immediately follows the inception $(4b)$ block and is made up of two convolutional layers using 128 "kernels of size 1×1", 128 "kernels of size $\#3 \times 3$ reduce", 256 "kernels of size 3×3", 24 kernels of size $\#5 \times 5$ reduce and 64 kernels of size 5×5. The output volume is of "size $14 \times 14 \times 512$".

Layers	Description
Layer 14 and 15	Inception $(4d)$ block follows the inception $(4c)$ block and is made up of two convolutional layers using 112 "kernels of size 1×1", 144 "kernels of size $\#3 \times 3$ reduce", 288 "kernels of size 3×3", 32 kernels of size $\#5 \times 5$ reduce and 64 kernels of size 5×5, resulting in a volume of "size $14 \times 14 \times 528$".
Layers 16 and 17	The inception $(4d)$ is immediately followed by the inception $(4e)$ block and is made up of two convolutional layers using 256 "kernels of size 1×1", 160 "kernels of size $\#3 \times 3$ reduce", 320 "kernels of size 3×3", 32 kernels of size $\#5 \times 5$ reduce and 128 kernels of size 5×5, resulting in a volume of "size $14 \times 14 \times 832$". The output from inception $(4e)$ is then fed as input to a maximum pooling layer which uses a patch size of 3×3 at a stride of 2 to downsample the input, giving an output volume of $7 \times 7 \times 822$.
Layers 18 and 19	The output from the pooling layer is further fed to layers 18 and 19 which are referred to as inception $(5a)$. Inception $(5a)$ is made up of two convolutional layers using 256 "kernels of size 1×1", 160 "kernels of size $\#3 \times 3$ reduce", 320 "kernels of size 3×3", 32 kernels of size $\#5 \times 5$ reduce and 128 "kernels of size 5×5", resulting in a volume of "size $7 \times 7 \times 832$".
Layers 20 and 21	These layers are the final inception blocks and are called inception $(5b)$. Similar to all other inception blocks, inception $(5b)$ is made up of two convolutional layers using 384 "kernels of size 1×1", 192 "kernels of size $\#3 \times 3$ reduce", 384 "kernels of size 3×3", 48 kernels of size $\#5 \times 5$ reduce and 128 "kernels of size 5×5", resulting in a volume of "size $7 \times 7 \times 1024$". The output from inception $(5b)$ fed to an average pooling layer. This pooling layer is the only one where the average pooling is used, and pooling is done with a patch size of 7×7 using a stride of 1, thus giving an output volume of $1 \times 1 \times 1024$.
Layer 22	This is the final layer of GoogLeNet. Here a dropout regularization of 40% is used in a fully connected layer, with 1000 neurons. A softmax function is applied after the fully connected layer.

Table 3.3: Description of GoogLeNet architecture

3.2.3.1 Inception ResNet V2

The Inception ResNet is a revised version of GoogLeNet. The Inception ResNet was inspired by the performance of the ResNet which made the authors of GoogLeNet propose a hybrid of Inception and ResNet to further boost performance. The Inception ResNet is comprised of a stem block, three Inception modules (A,B and C) and two reduction blocks (A and B). Figure 3.14 shows the configuration of the Inception Cells A, B and C.

The authors of Inception decided to introduce residual connections which aggregates the result from the convolution performed at the inception cell to the input. However, for this to work, the input and output must have the same dimensions, thus, a 1×1 convolution is done after each major convolution in order to make the depth the same. The authors further replaced the pooling operation with the residual connections, however, maximum pooling is still done inside the Reduction blocks. Figure 3.15 shows the Reduction blocks A and B.

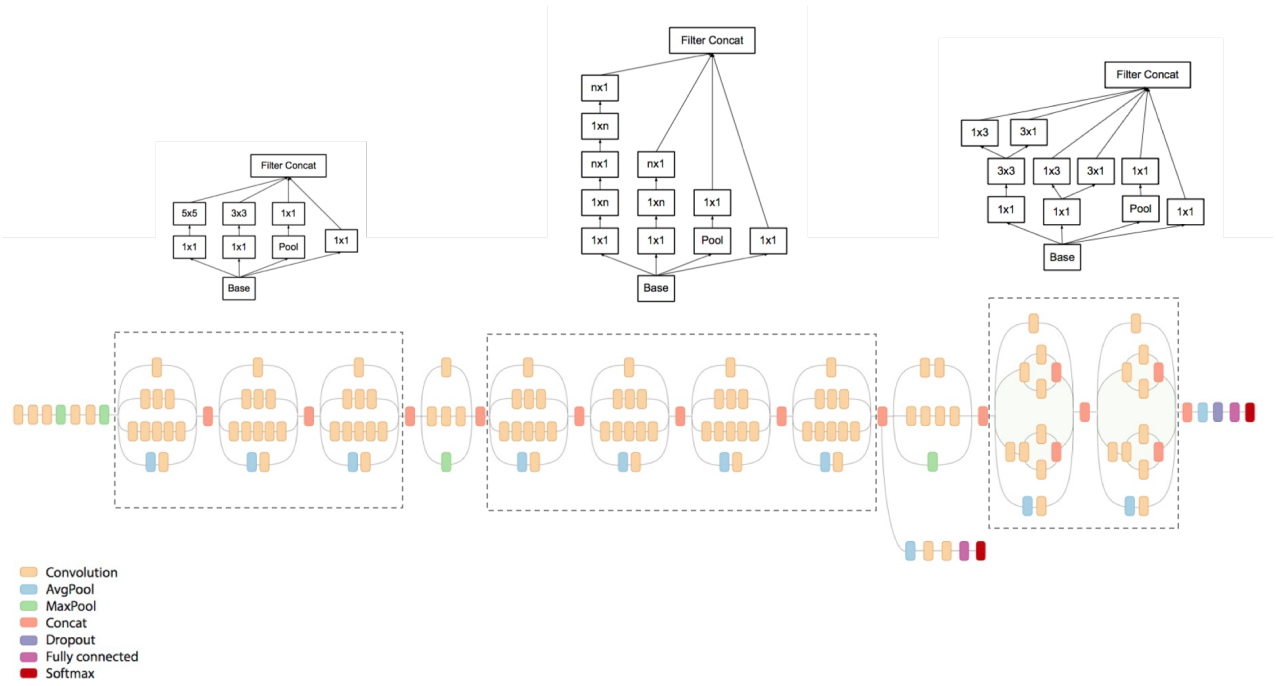


Figure 3.13: Revised GoogLeNet architecture) [7]

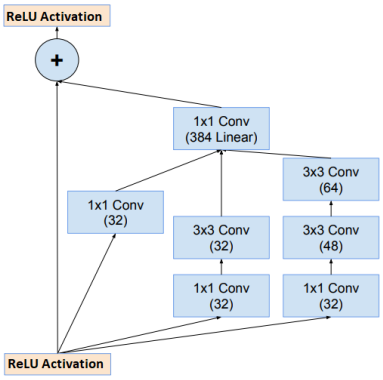
If the number of kernels exceed 1000 in networks with residual units, the residual variants tend to exhibit insecurities leading to "death" of the network, which means that the last layer before the average pooling (see Figure 3.16) starts returning only zeros after some iterations, and unfortunately, an addition of batch normalization does not resolve this problem nor lowering the learning rate does. However, in order to stabilize the training, the authors scaled the residual activations to a value between 0.1 and 0.3. Figure 3.16 shows the schema for Inception ResNet V2.

3.2.4 VGGNet

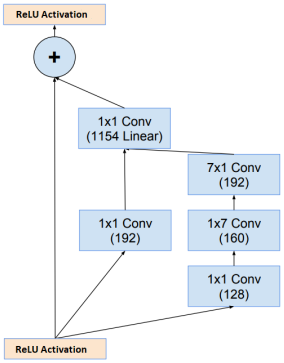
The Visual Geometry Group Network (VGGNet) architecture is a very simple yet powerful convolutional neural network architecture. The architecture came to be when Karen Simonyan and Andrew Zisserman [107] made a research into how the depth of a convolutional neural network affects the overall accuracy of the model in the large scale image recognition setting. Karen and Andrew increased the depth of their architecture to 16 and 19 layers, they also used very small kernels (of "size 3×3"); the two architectures (16 and 19 layers) are thus called VGG 16 and VGG 19 respectively.

The diagram shown in Figure 3.17 describes the architecture of the 16 layered neural network.

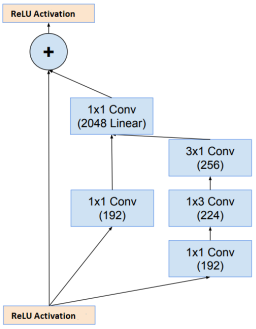
The VGG Network is notable for its high performance and it was the first runner up in the ImageNet challenge (image classification task) in 2014, although, it won the in localization task in ILSVRC 2014. In this research, we focus on the VGG 16 architecture.



(a) Inception module A [8]

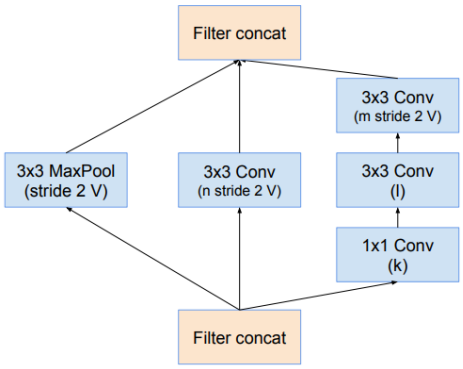


(b) Inception module B [8]

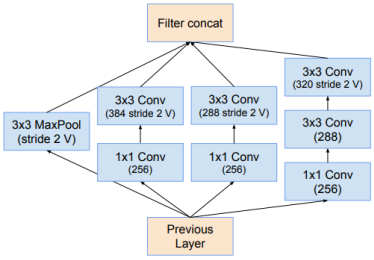


(c) Inception module C [8]

Figure 3.14: Inception modules in Inception ResNet V2 [8]



(a) Reduction block A - 35×35 to 17×17 grid reduction [8]



(b) Reduction block B - 17×17 to 8×8 grid reduction [8]

Figure 3.15: Reduction blocks in Inception ResNet V2 [8]

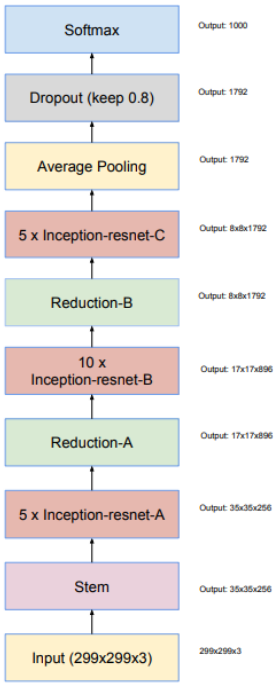


Figure 3.16: Schema for Inception ResNet V2 [8]

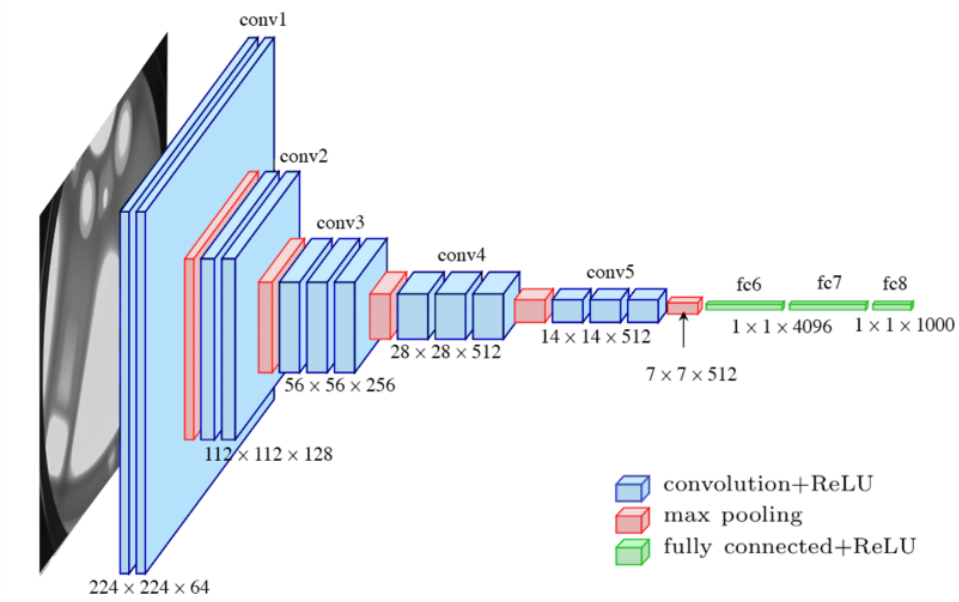


Figure 3.17: VGG configurations. The depth of the configurations increases (with increase in layers) from A to E) [9]

3.2.4.1 VGG-16

The VGG 16 network architecture comprises of 13 convolutional layers and three fully connected (FC) layers, thus, earning it the name VGG-16.

The VGG-16 architecture consists of 16 layers excluding the input layer. The layers and their operations are discussed in Table 3.4.

Layers	Description
Layers 1 and 2	The input to the network is an image x_i of "size $224 \times 224 \times 3$". The image is first fed to the first two convolutional layers 1 and 2 where we convolve with 64 "kernels of size 3×3" using SAME pooling with a stride of 14. The dimension of the image thus becomes $224 \times 224 \times 64$ which is then fed to the pooling layers which uses maximum pooling with a patch size of 3×3 at a stride of 2. The output of dimension of the image will thus be reduced to $112 \times 112 \times 64$. **Activation function: ReLU.**
Layers 3 and 4	In these layers, we have two convolutional layers with 128 "kernels of size 3×3" applied with a stride of 1. The image is then fed into a maximum pooling layer which uses a patch size of 3×3, thus reducing the input to $56 \times 56 \times 128$. **Activation function: ReLU.**
Layers 5 and 6	In these layers, 256 "kernels of size 3×3" are used for convolution. We then apply maximum pooling using a patch size of 3×3 with a stride of 2. The resulting output here is thus $28 \times 28 \times 256$. **Activation function: ReLU.**
Layers 7,8,9,10,11 and 12	There are three pairs of convolutional layers here and each paid is immediately followed by maximum pooling. The convolutional layers have 512 "kernels of size 3×3" using a stride of 1 and the maximum pooling layer with kernel size of 3×3 with a stride of 2 using 512 kernels. The final resulting image dimension is thus $7 \times 7 \times 512$. **Activation function: ReLU.**
Layer 13	In this layer, the output from the previous convolutional layer is now flattened through a fully connected layer thus having 25,088 units. **Activation function: ReLU.**
Layer 14 and 15	In these layers, we have two fully connected layers having 4096 units.
Layers 16	This is the last layer and it is a softmax output layer which has 1000 possible values. **Activation function: Softmax**

Table 3.4: Description of VGGNet architecture

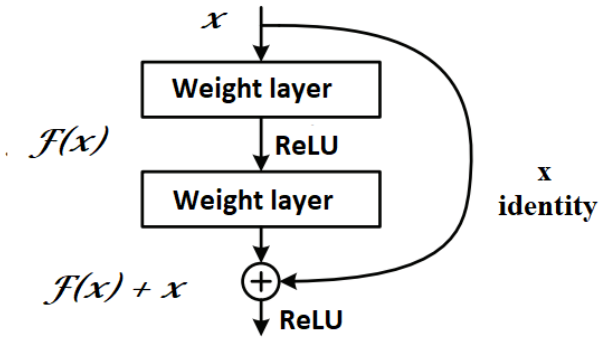


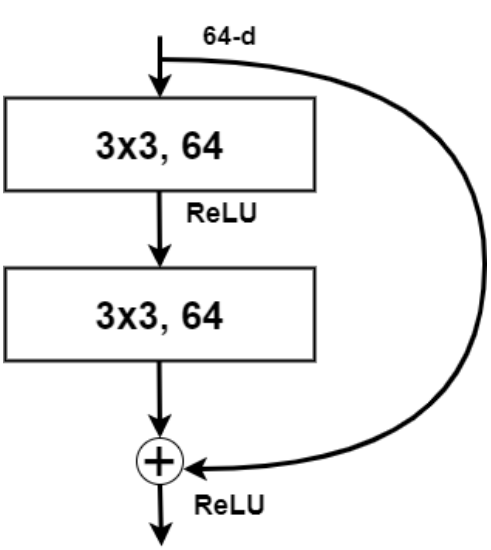
Figure 3.18: Residual learning: a building block. [10]

3.2.5 ResNet

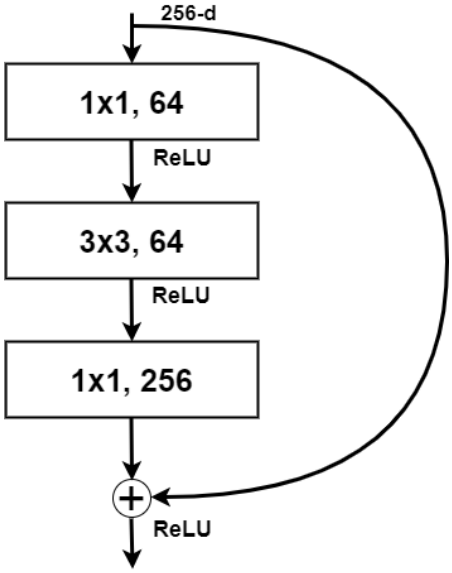
The residual network architecture shortened ResNet was introduced by Kaiming et al., (2015). The ResNet is a variant of convolutional neural network which won the ILSRVC in 2015. The ResNet is famously known for its high performance and beating human error rate of 5.1% with its own low error rate of 3.57%. A general principle of deep learning is that neural networks are capable of learning more complex functions of an input and should thus be able to yield better performance, however, researchers found out that adding more layers eventually had a negative contribution towards the overall performance. The developers of the ResNet architecture observed that this is due to what is called "vanishing gradients" or "degradation", so they proposed the residual network which maintains the gradients. Vanishing gradients majorly occurs in neural networks which have a high number of layers because we use back propagation to update the weights, thereby increasing the difficulty in updating the first layers of the network. The reason for this difficulty is that the gradient error gets reduced to zero by the time it reaches the first layers due to the chain rule which multiplies the error gradient values lower than one. The residual network introduces what is called "residual blocks" in which intermediate layers of a block learn a residual function with reference to the block input. Generally, the input matrix calculates in two linear transformations with the ReLU activation function; however, in ResNet, what happens is that it copies the input matrix directly to the second transformation output and then aggregate the output in final ReLU function.

There are several variants of the ResNet architecture of which include ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152. The ResNet 34 block (which is 2 layer deep) and ResNet 50 block (which is 3 layer deep) architectures are shown in Figure 3.19. The ResNet-50 model is similar to the ResNet-34, however, the authors replaced each two layer residual block with a three layer bottleneck block which makes use of 1×1 convolutions in order to reduce the depth of the channel thereby reducing the cost of computation when processing for 3×3 convolution. The ResNet-152 architecture (which at the time of its introduction was the deepest network ever presented on ImageNet) comprises of 152 layers with skip connections also referred to as gated recurrent units) and also features heavy batch normalisation, and despite its depth, it still achieved lower complexity than its counterpart VGGNets.

The ResNet developers observed that there would be better performance if the gradients are allowed to be propagated through the network during training. The diagram in Figure 3.20 shows an example architecture of the residual network comparing the network to VGG-19.



(a) ResNet-34 residual block



(b) ResNet-50 residual block

Figure 3.19: Residual blocks for the native and revised networks.) [10]

In Table 3.5, we briefly describe the 50 layers in ResNet-50 architecture.

Layers	Description
Layer 1	The ResNet layer starts with a convolutional layer. In this layer, a 224×224 image is convolved using 64 "kernels of size 7×7", using a stride of 2. The output from this convolution is of "size 112×112", which is then fed to a maximum pooling layer, using a patch size of 3×3 at a stride of 2. After pooling, the output volume is of size "56×56".
Layers 2, 3, 4, 5, 6, 7, 8, 9 and 10	These layers are referred to as a "residual layer". A residual layer is a combination of several blocks. A block here comprises of convolutional layers, which means a block can contain two or three convolutional layers. A two-layered block is called a "basic block", while a 3-layered block is called "bottleneck block". In this residual layer, we have three bottleneck blocks.
Layers 2, 3 and 4	The first bottleneck block comprises of convolutional layers 2, 3 and 4. In layer 2, we convolve the output from layer 1, which is 112×112, with 64 "kernels of size 1×1", then feed the output to layer 3 where we convolve 64 "kernels of size 3×3", and the result from layer 3 is fed to convolutional layer 4, where we convolve using a 256 "kernels of size 1×1".
Layers 5, 6 and 7	The second bottleneck block comprises of convolutional layers 5, 6 and 7. This bottleneck block is exactly similar in operation to the first bottleneck block, in that we use 64 "kernels of size 1×1", 64 "kernels of size 3×3", and 256 "kernels of size 1×1" in layers 5, 6 and 7 respectively.

Layers	Description
Layers 8, 9 and 10	The third bottleneck block which is the last in this residual layer comprises of convolutional layers 8, 9 and 10. This bottleneck block is also similar in operations to the other two blocks in this residual layer and use 64 "kernels of size 1×1", 64 "kernels of size 3×3", and 256 "kernels of size 1×1" in layers 8, 9 and 10 respectively. The output from this residual layer is of size "56×56".
Layers 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 and 22	These layers are another residual layer, these layers are the second residual layer in the ResNet-50 network. This residual layer comprises of 4 bottleneck blocks, and the first bottleneck block is where we downsample the input volume.
Layers 11, 12 and 13	The first bottleneck block in this residual layer comprises of 3 convolutional layers (11, 12 and 13), and in convolutional layer 11, we downsample the input volume of 56×56 using 128 "kernels of size 1×1" at a stride of 2. The output which is now reduced, is fed to layer 12, where we convolve using 128 "kernels of size 3×3", the output volume from layer 12 is then fed to layer 13 which is the final convolutional layer in the first bottleneck block. In layer 13, we use 512 "kernels of size 1×51".
Layers 14, 15 and 16	The second bottleneck block comprises of convolutional layers 14, 15 and 16. This bottleneck block is exactly similar in operation to the first bottleneck block, except that in layer 14, we do not downsample anymore, rather, we convolve using 128 "kernels of size 1×1", 128 "kernels of size 3×3", and 512 "kernels of size 1×1" in layers 14, 15 and 16 respectively.
Layers 17, 18 and 19	The third bottleneck block in this residual layer comprises of convolutional layers 17, 18 and 19. This bottleneck block is also similar in operations to the second block in this residual layer. In this block, we use 128 "kernels of size 1×1", 128 "kernels of size 3×3", and 512 "kernels of size 1×1" in layers 17, 18 and 19 respectively.
Layers 20, 21 and 22	The fourth bottleneck block which is the last in this residual layer comprises of convolutional layers 20, 21 and 22. This bottleneck block is also similar in operations to the second and third blocks in this residual layer. In this block, we use 128 "kernels of size 1×1", 128 "kernels of size 3×3", and 512 "kernels of size 1×1" in layers 20, 21 and 22 respectively. The output from this residual layer is 28×28.
Layers 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39 and 40	These layers are another residual layer, these layers are the third residual layer in the ResNet-50 network. This residual layer comprises of 6 bottleneck blocks, and the first bottleneck block is where we downsample the input volume.
Layers 23, 24 and 25	The first bottleneck block in this residual layer comprises of 3 convolutional layers (23, 24 and 25), and in convolutional layer 23, we downsample the input volume of 28×28 using 256 "kernels of size 1×1" at a stride of 2. The output which is now reduced, is fed to layer 24, where we convolve using 256 "kernels of size 3×3", the output volume from layer 24 is then fed to layer 25 which is the final convolutional layer in the first bottleneck block. In layer 25, we use 1024 "kernels of size 1×51".

Layers	Description
Layers 26, 27 and 28	The second bottleneck block comprises of convolutional layers 26, 27 and 28. This bottleneck block is exactly similar in operation to the first bottleneck block, except that in layer 26, we do not downsample anymore, rather, we convolve using 256 "kernels of size 1×1", 256 "kernels of size 3×3", and 1024 "kernels of size 1×1" in layers 26, 27 and 28 respectively.
Layers 29, 30 and 31	The third bottleneck block in this residual layer comprises of convolutional layers 29, 30 and 31. This bottleneck block is also similar in operations to the second block in this residual layer. In this block, we use 256 "kernels of size 1×1", 256 "kernels of size 3×3", and 1024 "kernels of size 1×1" in layers 29, 30 and 31 respectively.
Layers 32, 33 and 34	The fourth bottleneck block in this residual layer comprises of convolutional layers 32, 33 and 34. This bottleneck block is also similar in operations to the second and third blocks in this residual layer. In this block, we use 256 "kernels of size 1×1", 256 "kernels of size 3×3", and 1024 "kernels of size 1×1" in layers 32, 33 and 34 respectively.
Layers 35, 36 and 37	The fifth bottleneck block in this residual layer comprises of convolutional layers 35, 36 and 37. This bottleneck block is also similar in operations to the second, third and fourth blocks in this residual layer. In this block, we use 256 "kernels of size 1×1", 256 "kernels of size 3×3", and 1024 "kernels of size 1×1" in layers 35, 36 and 37 respectively.
Layers 38, 39 and 40	The sixth bottleneck block which is the last in this residual layer comprises of convolutional layers 38, 39 and 40. This bottleneck block is also similar in operations to the all the other blocks in this residual layer except for the first block. In this block, we use 256 "kernels of size 1×1", 256 "kernels of size 3×3", and 1024 "kernels of size 1×1" in layers 38, 39 and 40 respectively. The output from this residual layer is of "size 14×14".
Layers 41, 42, 43, 44, 45, 46, 47, 48 and 49	These layers are the final residual layer in the ResNet-50 network. This residual layer comprises of 3 bottleneck blocks, and the first bottleneck block is where like other residual layers, we downsample the input volume.
Layers 41, 42 and 43	The first bottleneck block in this residual layer comprises of 3 convolutional layers (41, 42 and 43), and in convolutional layer 41, we downsample the input volume of 14×14 using 512 "kernels of size 1×1" at a stride of 2. The output which is now reduced, is fed to layer 42, where we convolve using 512 "kernels of size 3×3", the output volume from layer 42 is then fed to layer 43 which is the final convolutional layer in the first bottleneck block. In layer 43, we use 2048 "kernels of size 1×51".
Layers 44, 45 and 46	The second bottleneck block comprises of convolutional layers 44, 45 and 46. This bottleneck block is exactly similar in operation to the first bottleneck block, except that in layer 44, we do not downsample anymore, rather, we convolve using 512 "kernels of size 1×1", 512 "kernels of size 3×3", and 2048 "kernels of size 1×1" in layers 44, 45 and 46 respectively.
Layers 47, 48 and 49	The third bottleneck block is the final block in this residual layer and it comprises of convolutional layers 47, 48 and 49. This bottleneck block is also similar in operations to the second block in this residual layer. In this block, we use 512 "kernels of size 1×1", 512 "kernels of size 3×3", and 2048 "kernels of size 1×1" in layers 47, 48 and 49 respectively. The output from this residual layer is of "size 7×7". The 7×7 output is then fed to an average pooling layer.

Layers	Description
Layer 50	This layer is the final layer, and is called fully connected layer. This layer uses the softmax activation.

Table 3.5: Description of ResNext architecture

The revision to the residual network is shown in Figure 3.21.

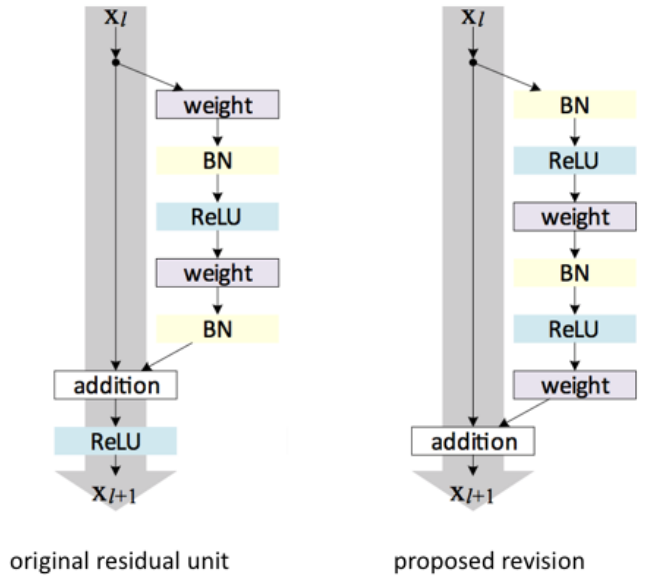


Figure 3.21: Native residual unit and revised residual unit) [10]

Several other variants of ResNet have been proposed which have also achieved considerable improvement in performance such as the ResNext-50. In this research, we will be using the ResNext-50 variant.

3.2.5.1 ResNext-50

The ResNext-50 is a variant of the ResNet which was proposed by Xie et al., (2017) [12], which is similar to the ResNet-50, except that the authors of ResNext modified the building block of the original ResNet as shown in the Figure 3.22.

The building block is similar to the inception cell of GoogLeNet described in section 3.2.3, in that, the ResNext building block and Inception cell both follow a split-transform-merge procedure, however, unlike in Inception cell where the output of the paths are merged through depth-concatenation, in ResNext's building block, the

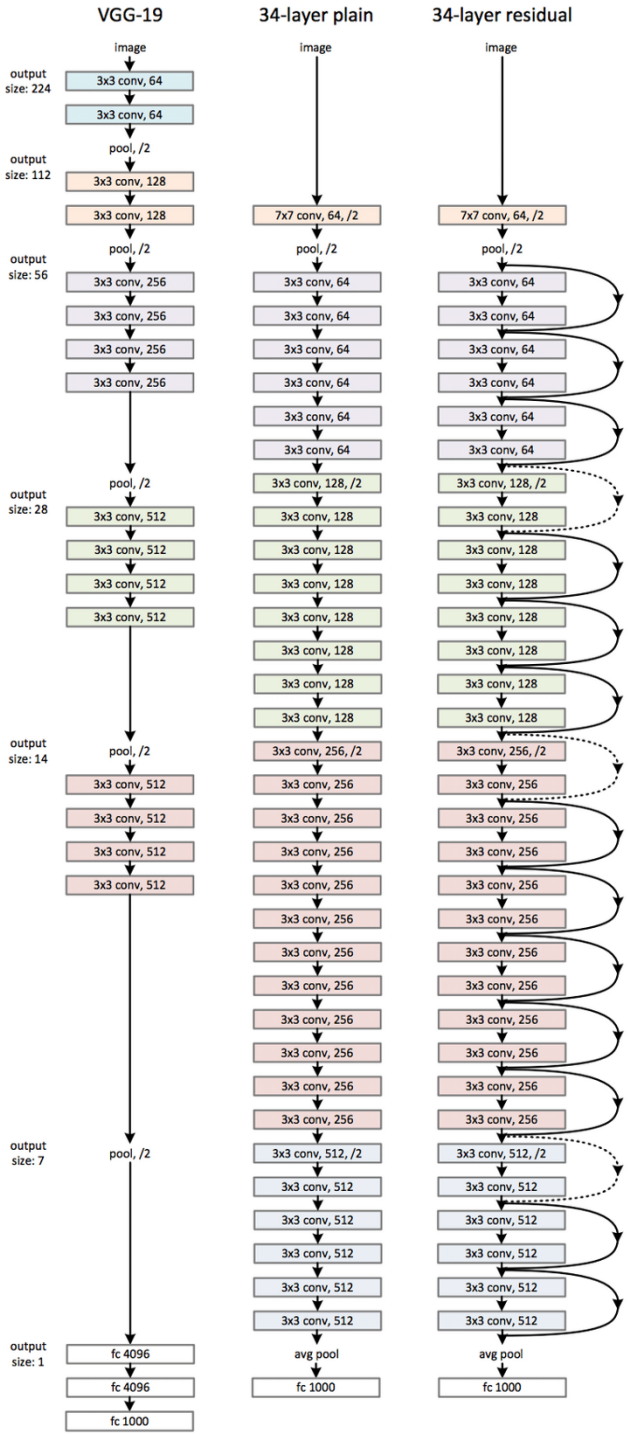


Figure 3.20: Comparing example residual network architecture for ImageNet. The left architecture is a VGG-19 model, middle architecture is a 34 parameter layer plain network and the right architecture is a 34 parameter layer residual network.) [11]

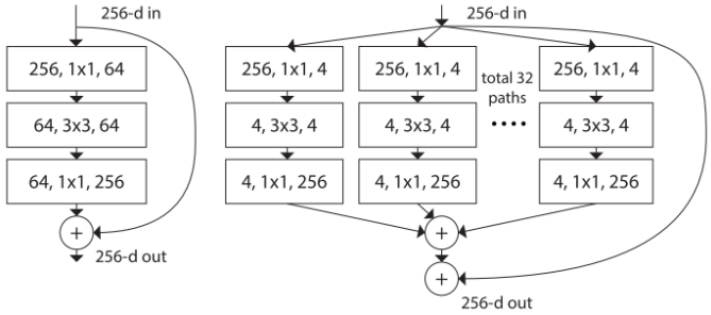


Figure 3.22: Left: Building block of ResNet; Right: Building block of ResNext with a cardinality of 32) [12]

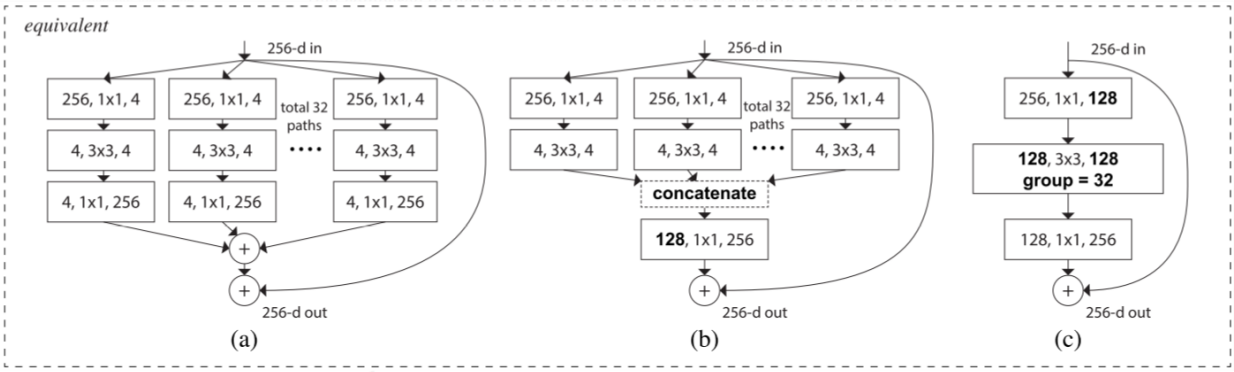


Figure 3.23: **(a)**: Aggregated residual transformation (same figure as 3.22). **(b)**: Building block equivalent to (a), implemented as early concatenation, **(c)**: Building block equivalent to (a) and (b), implemented as grouped convolutions. The bold part in (b) and (c) shows the reformulation changes. [12]

output of different paths are merged by adding them together and also, ResNext's variant features same topology in all merged paths, whereas, Inception cell features different topology for the merged paths (such as 1×1, 3×3, and 5×5 convolution. In ResNext, the authors introduced a hyper-parameter called "Cardinality" in order to allow for the adjustment of the model's capacity. Unlike in Inception architecture where there are several hyperparameters to be adjusted, the ResNext's model only has the cardinality hyperparameter to be adjusted thus making the network to more easily adapt to new data. Experiments show that to improve the performance of the model, we can rather increase cardinality instead of going deeper or wider. The relationship between the ResNext building block, Inception ResNet block and AlexNet grouped convolutions are shown in Figure 3.23.

3.3 Face Detection and Alignment using Multi Task Cascaded Convolutional Neural Network

The Multi Task Cascaded Convolutional Neural Network (MTCNN) was proposed by Kaipeng Zhang et. al in their 2016 paper [91].

The MTCNN is a cascade of three convolutional neural networks (CNNs) called the Proposal Network (P-Net), Refinement Network (R-Net) and the Output Network (O-Net). An RGB image x_i is firstly resized into different scaled to build a kind of image pyramid. This image pyramid serves as the input to the three-staged cascaded framework.

The Proposal Network is firstly used to generate candidate facial windows and their bounding box regression vectors. The candidate windows are then calibrated based on the estimated bounding box regression vectors. The candidate windows which highly overlap are then merged using Non-Maximum Suppression (NMS). The learning objective in the first instance is formulated as a binary classification problem. The loss function used to optimize the training on a data sample x is the cross entropy loss function which is mathematically described as:

$$L_i^c(x_i) = -(y_i^c \log(P_i(x_i)) + (1 - y_i^c)(1 - \log(p_i(x_i))), \tag{3.9}$$

where c implies the classification task, that is, L_i^c refers to the classification loss and $y_i^c \in \{0, 1\}$ is the classification ground truth label - face or not face, and $p_i(x_i)$ is the probability that the data sample x_i is a face.

The second stage of the MTCNN is the Refinement process. The candidate windows are fed into the Refinement Network (R-Net) which again discards a large number of false candidates. The NMS again is used to merge highly overlapping candidate windows which have been calibrated with the bounding box regression vectors.

The bounding box regression task is formulated as predicting the offset between a candidate window and the nearest actual values are the bounding boxes' top left x and y coordinates, and the height and the width of the bounding box. The learning objective is formulated as a regression task and the euclidean loss is used to calculate the offset for a given data sample x_i. The euclidean loss can be mathematically described as:

$$L_i^B = ||\hat{y}_i^B - y_i^B||_2^2, \tag{3.10}$$

where B refers to the bounding box regression task, that is, L_i^B, is the euclidean loss of the regression task, $\hat{y}_i^B$ is the predicted continuous value produced by the network and y_i^B is the actual continuous value (for the regression task).

$$\{y_i^B, \hat{y}_i^B\} \in \mathbb{R}^4. \tag{3.11}$$

The final network called the Output Network (O-Net is similar to the Refinement Network, however,in this final stage, the focus or objective is to identify the face regions with more supervision. The output network wold produce a total of ten values which correspond to the x and y coordinates of five locations on the face namely: left eye, right eye, nose, left lip end and right lip end.

The facial landmark localization problem is also formulated as a regression problem and we minimize the loss using Euclidean loss.

$$L_i^{lmark} = ||\hat{y}_i^{lmark} - y_i^{lmark}||_2^2, \tag{3.12}$$

where $lmark$ refers to the landmark regression task, that is, L_i^{lmark} is the landmark euclidean loss, y_i^{lmark} is the output coordinates produced by the network and y_i^{lmark} is the actual coordinate on the data sample (for the landmark regression task).

$$\{\hat{y}_i^{lmark}, y_i^{lmark}\} \quad \in \mathbb{R}^{10}. \tag{3.13}$$

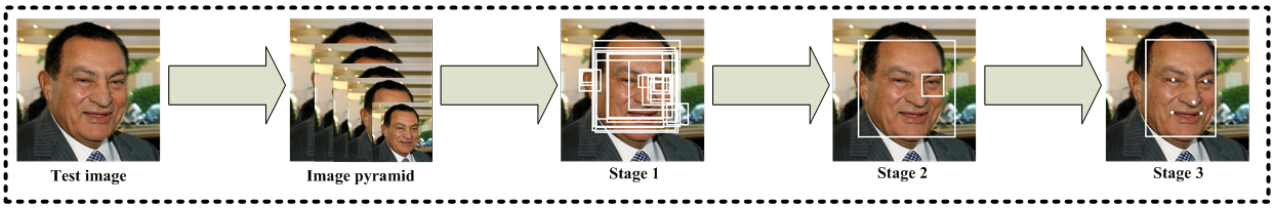


Figure 3.24: MTCNN architectural framework.
[91]

In some cases, while training the convolutional neural networks, there are different types of images such as partially aligned faces, face and not-face. In some of these instances, some of the loss functions used for the CNNs were not used. The overall learning objective of such scenarios is however described mathematically as:

$$\min \sum_{i=1}^{N} \sum_{j \in \{c,B,lmark\}} \alpha_j \beta_i^j L_i^j, \tag{3.14}$$

where N is the number of training samples andα_j is the importance of the task.

For the Proposal and Refinement Networks, the authors [91] set $\alpha_c = 1$, $\alpha_B = 0.5$, $\alpha_{lmark} = 0.5$, and

For the Output Network, the the authors [91] set $\alpha_c = 1$, $\alpha_B = 0.5$, $\alpha_{lmark} = 1$.

The authors further suggested using stochastic gradient descent to train the convolutional neural networks in special scenarios. The diagram shown in Figure 3.24 shows the pictorial representation of the stages involved in Multi Task Cascaded Convolutional Neural Networks (MTCNN) from first stage (Proposal Network) through the second stage (Refinement Network) till the third stage (Output Network).

3.4 Conclusion

In Section 3.1, we discuss convolutional neural networks in detail, including all the layers and operations done in convolutional neural networks. We also discussed deep convolutional neural networks which are just like conventional neural networks, but with deeper hidden layers. In section 3.2, we provided some convolutional neural network architecture variants such as LeNet-5, AlexNet, VGGNet, GoogLeNet, and ResNet. We further described some variants of some of these networks (which are the ones use in this research) such as VGG-16 for VGGNet, Inception ResNet V2 for Inception ResNet, and ResNext for ResNet and also detailed the operations performed in the different layers of each architecture. In Section 3.3, we introduce the multi task convolutional neural network (a CNN based faced detector) to show how CNNs have been applied to solve face detection problem in literature and this is a key segment of our research. The performance of deep convolutional neural networks in literature, especially the ones discussed in Section 3.2 inspires us to use deep CNNs to monitor driver alertness in real time.

Chapter 4

Research Methodology

4.1 Introduction

In Chapter 2, we provided a detailed explanation of how researchers have tried to tackle the problem of monitoring the state of drivers and warn them when they become drowsy. We established the context of the research as relating to solving this problem using visual information fed into machine learning algorithms. In Chapter 3, we discussed convolutional neural networks and several notable variants which have been known to perform well over the years. In this chapter, we provide the methodology for this research.

4.2 Overall Framework of the Driver Alertness Monitoring

Our driver alertness monitoring approach is a 3-staged framework. A graphical representation of the framework is described using the diagram in Figure 4.1.

Stage 1: An input image x_i, is fed into a face detection algorithm. For this research, we have chosen the Dlib as our default face detector after a careful analysis of its strengths and drawbacks which we discuss in detail in this chapter. Although, the Dlib HoG based face detector is a lightweight accurate face detector, it performs poorly on faces smaller than 80×80 and does not do very well with occlusion. We do expect that drivers will be close to the camera and the face will be reasonably larger than this size, but, we also expect occlusions, thus, to avoid any information loss, we have cascaded the face detector stage with the Multi task cascaded convolutional neural network (MTCNN) face detector which is CNN based. If the Dlib fails to capture a face in an image, the image is passed to the MTCNN which if unable to detect a face, the face is passed through to the MTCNN.

Stage 2: The face detectors produce a bounding box of the face and facial features coordinates. In this stage, we feed the faces produced in stage 1 to face-state detection algorithms which are different convolutional neural networks variants. The CNNs were trained on faces classified into three classes: “Drowsy”, “Normal” and “Yawning”. After training, the networks having learnt the differences between the three different face-states, are able to classify any given face into any of “Drowsy”, “Normal” or “Yawning”.

Stage 3: When a new face is fed to the algorithms in stage 2, the result (predicted face-state) is fed to the

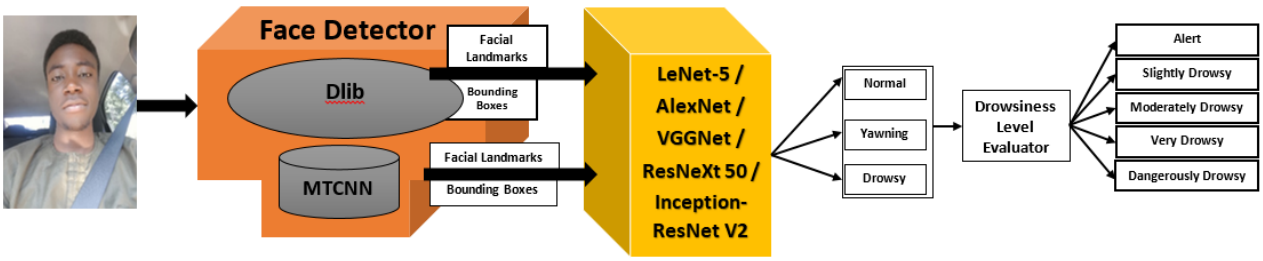


Figure 4.1: Overall architecture of the driver alertness monitoring system.

evaluator in stage 3. Although, the face-state "Drowsy" and "Yawning" are so called, the evaluator does not take one predicted drowsy face-state as "Drowsy", nor one predicted yawning face state as "Yawning", rather, it keeps track of a series of predictions, such that if there are T_y predictions within time t seconds, where T_y is the number of sequential "Yawning" face-state predictions and t is the average duration of a yawn, then, the evaluator penalizes with p points from the current alertness value (CAV), which is always initialized to 100%, as "Yawning". The output of the evaluator is the CAV which determines the present driver alertness level. When the CAV is at least 80%, the driver is deemed "Alert", while a CAV of 25% or less deems the driver as "Dangerously Drowsy". A CAV between 65% and 80% deems the driver as "Slightly Drowsy", a CAV between 50% and 65% deems the driver as "Moderately Drowsy", and a CAV between 26% and 50% deems the driver as "Very Drowsy". The framework stages are discussed in details in this chapter.

4.3 Data Collection

Developing a machine learning algorithm requires data to be trained on, however, deep learning models don't just need data, they need large data samples. Although, there are publicly available dataset, due to the nature of our deep learning architectures, using a single repository might not be suitable because we need a large dataset to train our algorithms. As a result, we combined several publicly available databases to form a single database for our system. Two major categories of data format we use for the training of our models are Images and Videos.

4.3.1 Images

In collecting image data, we focused on repositories which have people with their eyes closed or open or with their mouths wide open (yawning) or normal (talking or mouth closed). Although, there are a lot of databases which are available to monitor drowsiness, some are not suitable because of different factors such as angle, large bias to an ethnicity or race, presence of sensors or due to special permissions requirement and time taken to get permission from owners of some databases. After a careful consideration of these publicly available dataset, we selected the Closed Eyes in the Wild (CEW) database as it is readily and easily available for use.



Figure 4.2: Closed Eyes in the Wild image samples.
[112]

4.3.1.1 Closed Eyes in the Wild (CEW)

The CEW [108] database is a publicly available database consisting of closed and open eyes images of persons. The dataset consists of 2423 subjects with 1192 subjects with both eyes closed collected directly from Internet, and 1231 subjects with eyes open selected from the Labeled Face in the Wild database [109]. The dataset was collected by F. Song et al. for the purpose of investigating eye closure which is an herculean task in real world application scenarios due to individual differences [108]. The authors used the Viola and Jones algorithm [110] for face detection and eyes detection using eye localization approach in [111]. The dataset was pre-processed and the authors resized the faces to a size of 100×100 and extracted the eyes from these resized faces which are of sizes 24×24. Although, this dataset was created for eye closeness detection and not drowsiness detection, the dataset is still very much suitable to train a drowsiness detection system, because eye closeness or eye blinks are key factors in recognizing drowsiness. The diagram in Figure 4.2 shows sample images from the Closed Eyes in the wild database. Faces with eyes closed are shown in presented in the first row while faces with eyes open are presented in the second row.

4.3.1.2 Self Compiled Dataset

To have a larger dataset, we used the Google and Bing APIs to create additional custom dataset. We pulled images from these APIs using keywords such as "person sleeping", "person closed eyes", "person driving sleeping", "person yawning". The "person" keyword her represents: woman, man, lady, boy, girl and person. We also used other keywords like "presidents", "entrepreneurs" and "celebrities" among many others especially for the normal samples. Our custom data after filtering out the wrong images was a total of 400 yawning images, 453 normal images (largely from celebrities search), and 438 "drowsy" images. These data samples were then fed to the dlib face detectors, which as expected missed out on some faces, but, the faces missed were detected

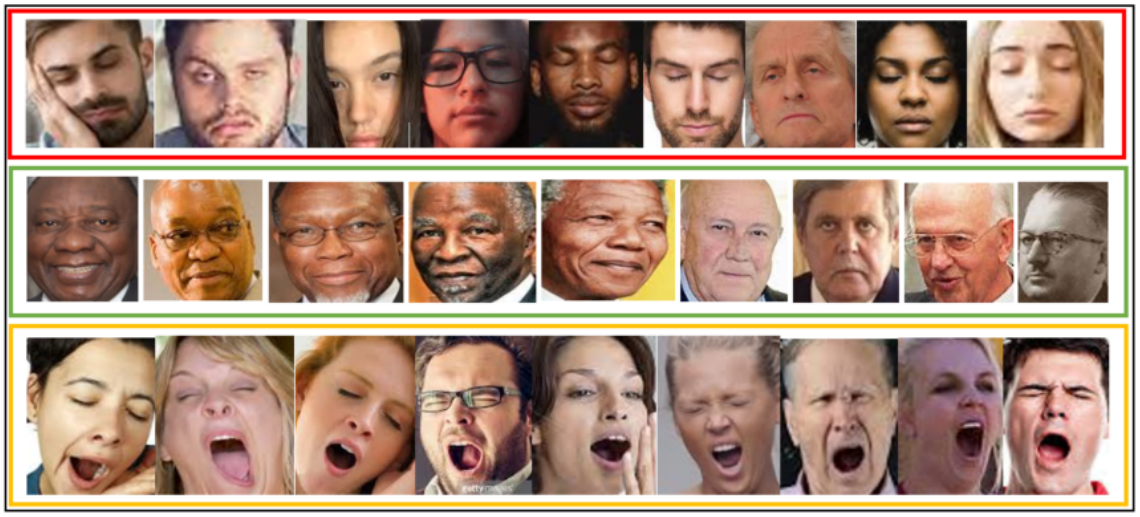


Figure 4.3: Samples of faces of images scraped from Google and Bing [113]

by the MTCNN, thus, giving a total of 400 yawning faces, 453 normal faces and 438 drowsy faces. Figure 4.3 shows samples of normal, drowsy and yawning faces from our compiled data. The images boxed in red are drowsy face samples, the images boxed in green are normal face samples and the images boxed in yellow are yawning face samples.

4.3.2 Videos

The CEW dataset provides data which are only grouped by us into normal and drowsy. There are thus few yawning samples, and to increase the yawning data samples, we opted for the YawDD dataset [114], which was used in [115]. The YawDD dataset contains videos of drivers participating in different actions such as talking, singing, laughing, yawning and closed mouthed.The dataset was collected to be used for testing algorithms and models for detecting yawns primarily but also secondarily for face and mouth tracking and detection. The YawDD dataset which consists of different two categories of videos, the first containing 322 videos of female and male drivers taken from a camera placed under the front mirror in a car. This category of videos feature people from different ethnicity and three different situations such as Normal driving or no talking, Talking or singing while driving and Yawning while driving. The second category of videos obtained from a camera installed under the driver's dash features 29 participants (male and female) with several combination of actions such as driving, driving while talking and driving while yawning. All videos are of AVI format and of "size 640×480" 24-bit true color recorded at 30 frames per second (fps). Further information about the YawDD dataset can be viewed at [114]. The videos were split into frames, and a total of 1176 frames were obtained from the yawning videos and a frame each where participant is drowsy and normal was extracted from each video, resulting in 356 drowsy samples and 356 normal samples. Figure 4.4 shows sample frames extracted from the videos in the YawDD dataset. The images in the blue boxes are samples from the camera installed under the front mirror while the images in red are the samples from the camera installed in the dash. The samples show male and female drivers from different ethnicity acting differently, and with varying facial additions such as

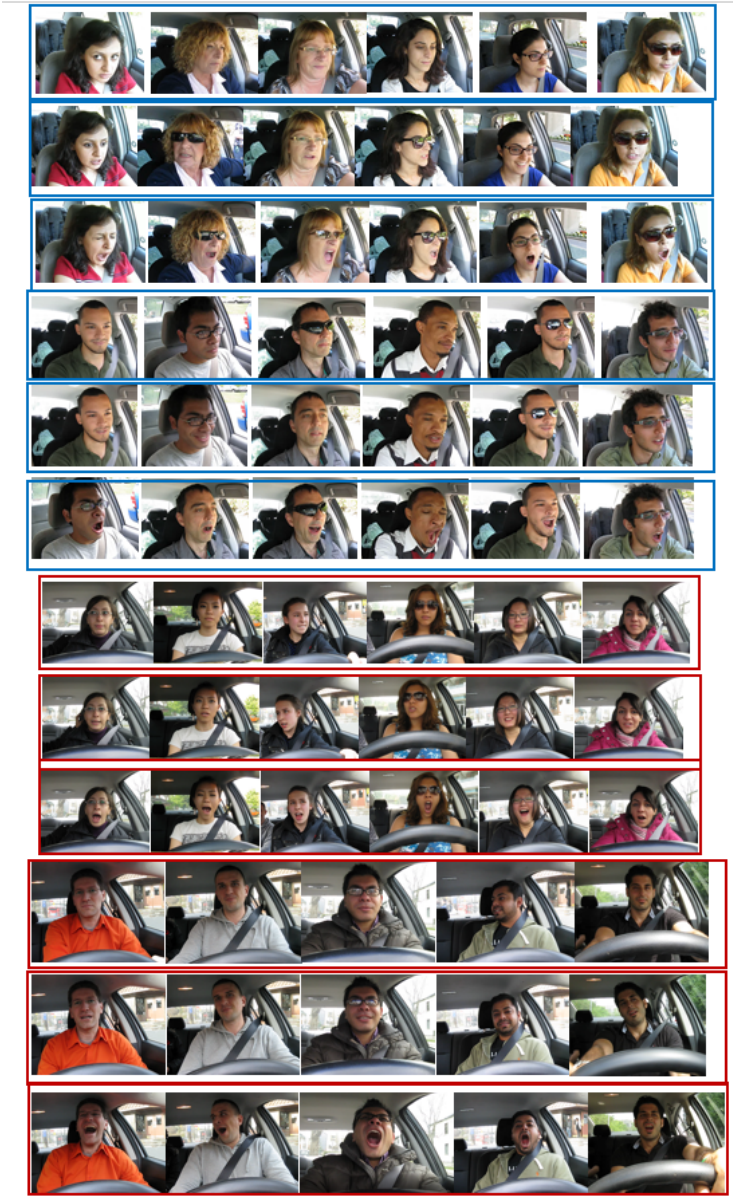


Figure 4.4: YawDD - Yawning Detection Dataset
[112]

use of prescribed glasses, sunglasses and having facial hair.

4.4 Data Preprocessing

We collected data from two publicly available databases: Closed Eyes in the Wild (CEW) [112] and YawDD [114]. We also scraped images through two search engines: Bing and Google. The YawDD, being a video dataset was first split into frames at 30 FPS. We filtered some frames belonging to other classes, for example, frames where participants are expected to be yawning but they were laughing. A total of 1888 frames were selected from the YawDD dataset. The dataset collected from CEW are strictly faces and were 2423 in total and the data samples compiled by us are 1291 in total.

4.4.1 Face Detection/Extraction

Face detection is the process of obtaining the bounding boxes of faces in an image. These bounding boxes tell us what part of there are human faces, and these bounding boxes coordinates can be used to extract only that part of the image.

There are several face detection algorithms based on different approaches, with different outputs. In this study, we considered five face detection algorithms, and our consideration is based on easy access to these face detectors which are also available as libraries in our preferred programming language (Python). The face detection algorithms we considered are discussed below:

4.4.1.1 Face Detection Using Haar Cascade (OpenCV)

The earliest face detection algorithm was proposed in 2001 by Paul Viola and Michael Jones[75] and this approach was very efficient at detecting faces in an image. Viola and Jones trained a cascade function on a large database of images containing faces and no faces at all. The output is a list of coordinates (x,y) containing 4 elements per face.

The main Advantages of OpenCV haar cascade face detector are as follows: 1.) Almost real-time face detection on CPU; 2.) Architecture is simple to design and understand; and 3.) Detects faces on different scales.

OpenCV haar cascade face detector also has some drawbacks and they include: 1.) High rate of false positives; 2.) Performs poorly on images that are not fully frontal; and 3.) Almost no detection when face is occluded.

4.4.1.2 Face Detection Using Deep Neural Networks (OpenCV)

This approach is based on Single Shot Detectors (SSD) [116] using the ResNet architecture [10]. The OpenCV SSD face detector is notable for its strengths which include: 1.) Very accurate face detection; 2.) Performs well under occlusion; 3.) Works well for different face orientations - up, down, side-face, right and left and does well on different scales of faces; and 4.) Detects faces in real-time on CPU.

4.4.1.3 Face Detection Using Histogram of Oriented Gradients (Dlib)

This approach uses Histogram of Oriented Gradients (HoG) features combined with Support Vector Machines (SVM) to detect faces in images. The model is built using 5 HoG filters which are: front looking, right looking, left looking, front looking with right rotation and front looking with left rotation. The Dlib HoG based face detector was trained with 2825 image samples collected from Labelled Faces in the Wild [109].

The Dlib HoG face detector has major strengths such as: 1.) Very fast face detection on CPU (fastest real time detection compared to others); 2.) Performs well under minimal occlusion; 3.) Detects frontal and slightly frontal faces; and 4.) Very lightweight model compared to the other face detectors. Although, the Dlib HoG has impressive strengths, it is not without some drawbacks. The drawbacks of Dlib HoG include: 1.) Does not detect small faces with pixels less than 80×80, this is majorly because it was trained for minimum face size of 80×80; 2.) Performs poorly on images with high occlusion; and 3.) Does not detect extremely non-frontal faces, for example, looking up or down.

4.4.1.4 Face Detection Using Deep Convolutional Neural Networks (Dlib)

This approach uses Maximum-Margin Object Detector (MMOD) with CNN based features. This approach is very simple and does not require very large amount of data to train. The model was trained on combined datasets from ImageNet [117], WIDER Face [118], PASCAL VOC [119], VGG-Face [120], and Face Scrub [121]. A total of 7220 images were used to train this deep learning based model. Dlib's CNN based face detector has major advantages which are as follows: 1.) Accurate detection of faces with different orientations; 2.) Fast detection on GPU; 3.) Very robust to occlusions; and 4.) Simple to train. The Dlib CNN based face detector however does not work in real time on CPU and it does not detect small faces with pixels less than 80×80, which is majorly because it was trained for minimum face size of 80×80.

4.4.1.5 Face Detection Using Multi Task Convolutional Neural Networks (MTCNN)

The MTCNN described elaborately in Section 3.3 is a state-of-the-art approach to face detection. The MTCNN returns a bounding box and 5 facial landmarks. The MTCNN does face alignment and detection together simultaneously, thus helping it to better detect faces which are not well aligned compared to the other face detectors listed above. The main advantages of MTCNN are as follows: 1.) Performs well on faces across

different scales; 2.) Accurate detection of faces; 3.) Real time computation on GPU; and 3.) High robustness to occlusion. The MTCNN however performs best on GPU.

4.4.1.6 Selection of Face Detection Algorithms

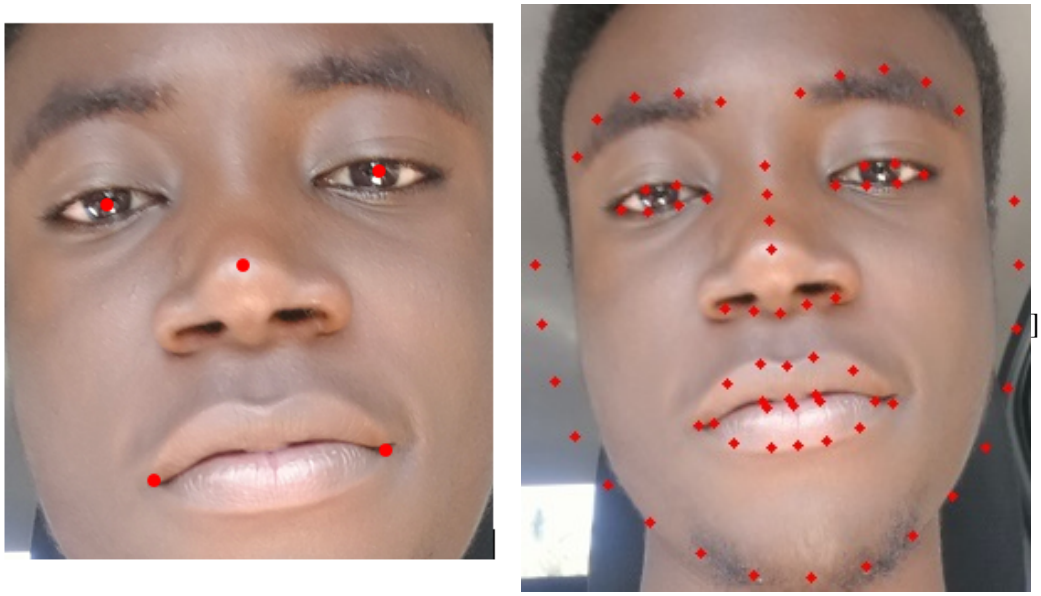
Selecting the best face detection algorithm suited for study is very important requires us to consider the strengths and weaknesses of the five face detection algorithms highlighted in this section. Our criteria to choose the best algorithm suited for our study are as follows:

1. The algorithm must be fast at detecting faces on CPU in shortest possible time. This is because, the time taken to detect faces will influence the overall time in monitoring drowsiness, and since our study aims to monitor drowsiness in real time, a fast computation is highly necessary on even a low computation capacity computer.

2. Accuracy: The face detector must be accurate at detecting faces. In our study, we want to avoid information loss as much as possible, and to minimize information loss, we need a face detector that detects faces well.

3. Robust to occlusion and detection of slightly tilted faces. We are aware that in real life scenarios, there would be occlusion such as use of glasses, hair covering part of the face, reflections, shadows or objects blocking a part of the face etc. Thus, we want our face detector to detect as much as possible in these scenarios.

Based on the criteria highlighted above, we observed that one face detector does not meet all requirements, thus, we prioritized the requirements and selected the best suited, which is the Dlib HoG based face detector. We chose the Dlib HoG based face detector because it is very fast at detecting faces on CPU. The Dlib HoG based face detector is the fastest among the 5 face detectors we discussed in this section. The Dlib also detects frontal and slightly frontal faces, however, does not detect extremely non-frontal faces. Two major drawbacks of the Dlib face detector are that it does not detect faces with pixels less than 80×80, and does not work well on images with occluded faces. We assume we need not worry about the first drawback as we expect that drivers would be very close to the camera and the face pixels would hardly be less than 80×80. The second drawback is a very concerning one which can not be compromised, and because of this, we include a second face detector, which is the MTCNN. We cascade the face detection stage, with the Dlib being the first face detector and the MTCNN, the second face detector. When an image x_i is fed to the face detector layer, the Dlib looks for a face and only if it is unable to detect one, image x_i is fed to the MTCNN. We chose MTCNN as our second face detector because of its high robustness to occlusion. The MTCNN is also very accurate at detecting faces which are not necessarily strictly frontal. Figure 4.5 shows our selected face detectors' output, given an input image x_i.



(a) MTCNN 5 Points Facial Landmarks

(b) Dlib 68 Points Facial Landmarks.

Figure 4.5: Selected Face detectors' Outputs.

4.4.2 Data Augmentation

The process of generating transformed versions of an original image in order to help increase the training samples is called "Data augmentation", and this technique helps models to be resistant against little changes which don't necessarily affect the image structure. In this research, we applied transformations using the Tensorflow Keras image data generator. The transformations applied to our image data are discussed below.

4.4.2.1 Vertical Shift Augmentation

A vertical shift augmentation means moving all the pixels in a vertical direction while maintaining the image dimension. A vertical shift thus causes some pixels along the vertical region to be cut off while new pixel values are added to the other side of the vertical region. Figure 4.6 shows different results of vertical shift augmentation on a face extracted from an image x_i.

4.4.2.2 Horizontal Shift Augmentation

An horizontal shift augmentation means moving all the pixels in an horizontal direction while maintaining the image dimension. Just like in vertical shift, some pixels are cut off, however only along the horizontal region and new pixel values are added to the other side of the horizontal region. Figure 4.7 shows different results of horizontal shift augmentation on a face extracted from image x_i.

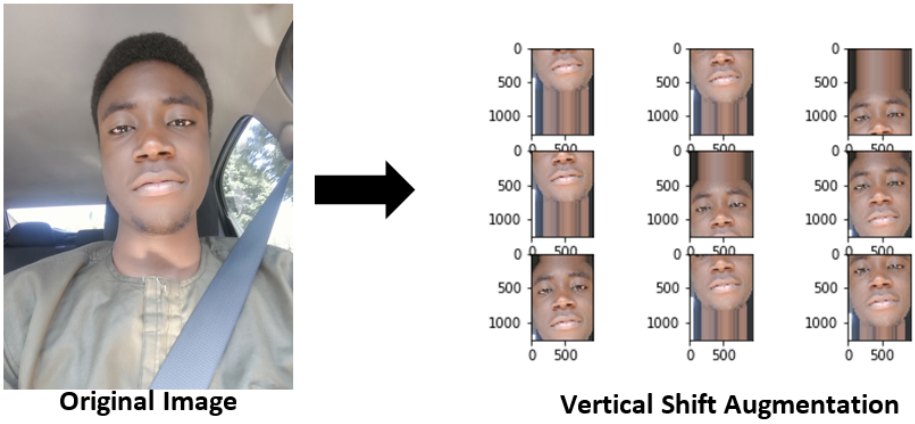


Figure 4.6: Vertical Shift Augmentation

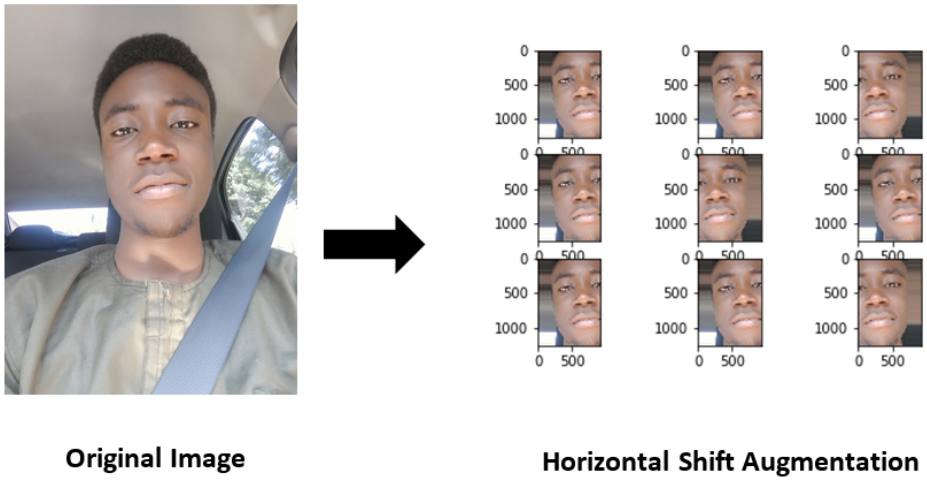


Figure 4.7: Horizontal Shift Augmentation

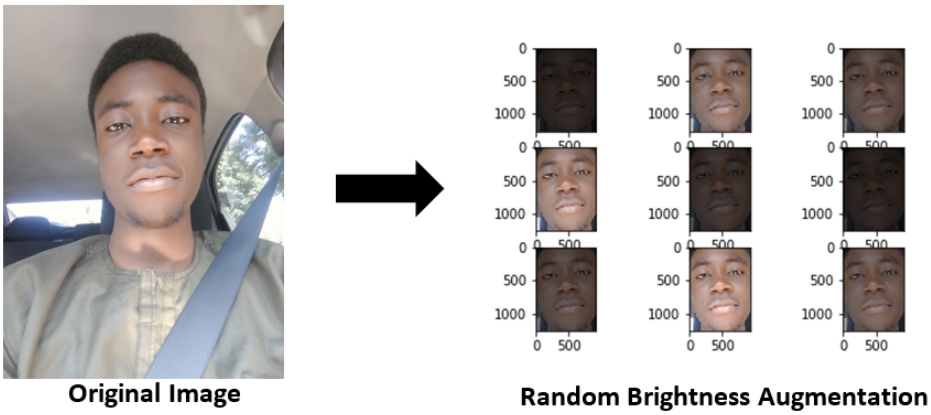


Figure 4.8: Random Brightness Augmentation

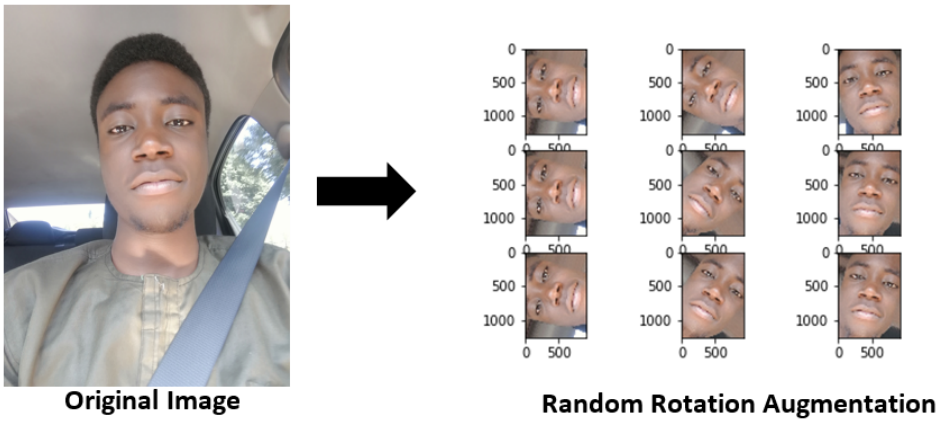


Figure 4.9: Random Rotation Augmentation

4.4.2.3 Random Brightness Augmentation

Random brightness means we randomly darken or brighten an image. Random brightness is a key data augmentation because it helps the model to generalize across different levels of brightness. Figure 4.8 shows different results of random brightness augmentation on a face extracted from an image x_i.

4.4.2.4 Random Rotation Augmentation

A random rotation means moving pixel values in a clockwise direction between 0 to 360 degrees. In this research, we have only used a rotation range of 90 degrees, which means pixel values are only rotated randomly between 0 and 90 degrees. Figure 4.9 shows different results of random rotation augmentation on a face extracted from image x_i.

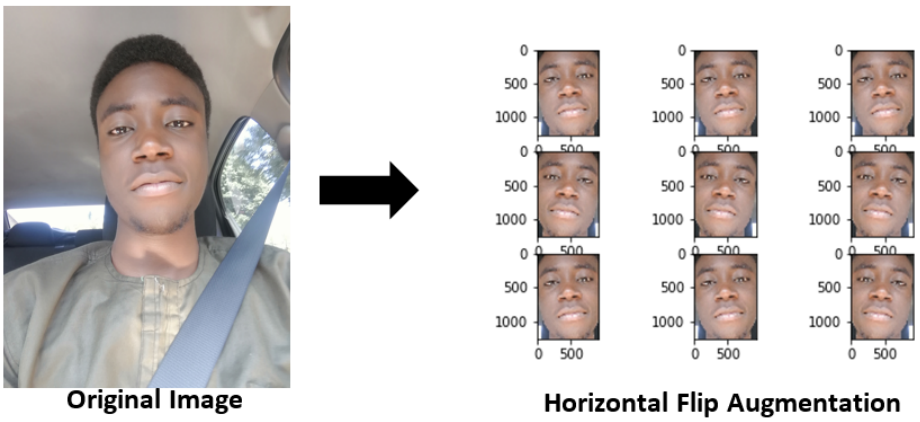


Figure 4.10: Horizontal Flip Augmentation

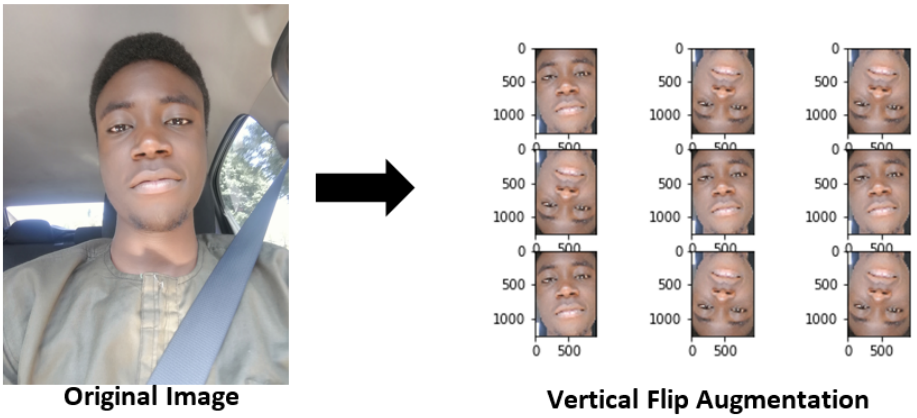


Figure 4.11: Vertical Flip Augmentation

4.4.2.5 Horizontal Flip Augmentation

An horizontal flip means we reverse the columns of pixels of an image. An horizontal flip can be understood as a mirrored image of an image. Figure 4.10 shows different results of horizontal flip augmentation on a face extracted from image x_i.

4.4.2.6 Vertical Flip Augmentation

Similar to an horizontal flip, in a vertical flip, we reverse the rows of pixels of an image. A vertical flip can be understood as turning an image upside down. Figure 4.11 shows different results of vertical flip augmentation on a face extracted from image x_i.

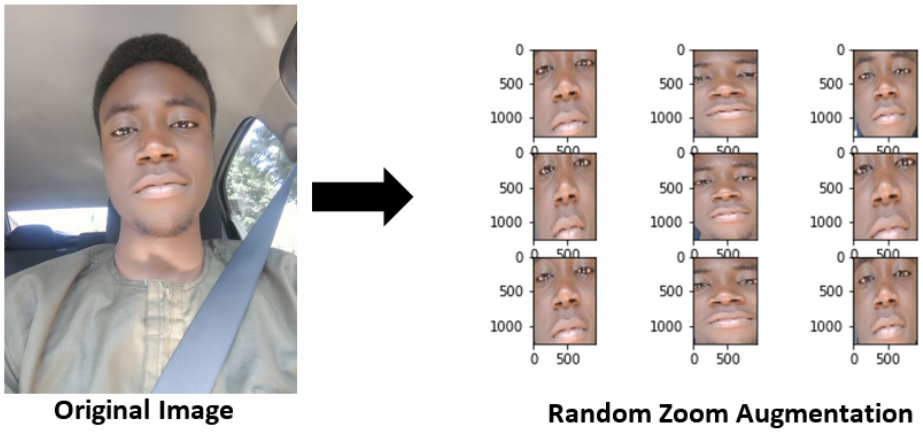


Figure 4.12: Random Zoom Augmentation

4.4.2.7 Random Zoom Augmentation

This data augmentation technique randomly zooms the image in and interpolate pixel values or add new pixel values around the image. Figure 4.12 shows different results of random zoom augmentation on a face extracted from image x_i.

4.5 Training

After preprocessing our data, we need to feed this data into our machine learning algorithms. In this research, we have selected 5 algorithms which are: LeNet, AlexNet, VGG-16, Inception ResNet V2 and ResNext-50. In this section, we discuss the configuration of the selected machine learning algorithms.

4.5.1 Architecture

Five convolutional neural network architecture variants were selected after a careful evaluation of their performances from literature. These architectures were trained with the same dataset, and where appropriate the data samples were resized according to the architecture of the network.

4.5.1.1 LeNet-5

LeNet-5 which is elaborately discussed in Section 3.2.1, is a CNN architecture which was introduced at a time computational capacity was limited. LeNet-5 accepts an input of 32×32 grayscale image I and performs

operations on the image as it moves from convolutional layer through to fully connected layer. To train our LeNet algorithm, we resized our facial images to $32 \times 32 \times 1$. We train the LeNet-5 on three classes of data: Normal (characterized by open eyes, closed mouth, mouth slightly open or talking), Yawning (majorly characterized by Mouth wide open and mostly eyes closed) and Drowsy (Eyes partially shut, totally closed eyes, face posed down).

4.5.1.2 AlexNet

AlexNet which is also discussed in Section 3.2.1, is another CNN architecture which is more complex than LeNet-5. We feed random crops of "size 227×227" of our training set using the data augmentation technique described in Section 4.3.3. This model as described in Chapter 3 primarily consists of 5 convolutional layers and 3 fully connected layers. We train the AlexNet on three classes of data: Normal (characterized by open eyes, closed mouth, mouth slightly open or talking), Yawning (majorly characterized by Mouth wide open and mostly eyes closed) and Drowsy (Eyes partially shut, totally closed eyes, face posed down).

4.5.1.3 VGG-16

We feed an input image of "size 224×224" to the VGG-16 which has a total of 138 million parameters. An important thing to note is that all the convolutional kernels are of "size 3×3" and the maximum pooling kernels are all of "size 2×2" with a stride of 2. The VGG16 consists of 13 convolutional layers (some of which are followed by maximum pooling layers) and three fully connected layers. Although there are 5 maximum pooling layers, making a total of 21 layers, the total number of weight layers is 16. We train the VGG-16 on three classes of data: Normal (characterized by open eyes, closed mouth, mouth slightly open or talking), Yawning (majorly characterized by Mouth wide open and mostly eyes closed) and Drowsy (Eyes partially shut, totally closed eyes, face posed down).

4.5.1.4 Inception ResNet V2

We feed an input image of "size $299 \times 299 \times 3$" to the Inception ResNet model which comprises of Inception Cells A,B and C, and two reduction blocks A and B. The Inception Cell A, is immediately followed by a Reduction Block A, followed by Inception Cell B, which is followed by another Reduction Block B. The output from the Reduction Block B is fed to another Inception Cell C which is later pooled using Average Pooling. We use a dropout regularization to make the network robust to overfitting, and finally feed to a 3 softmax classifier output layer. We train the Inception ResNet on three classes of data: Normal (characterized by open eyes, closed mouth, mouth slightly open or talking), Yawning (majorly characterized by Mouth wide open and mostly eyes closed) and Drowsy (Eyes partially shut, totally closed eyes, face posed down).

4.5.1.5 Residual Network (ResNext-50)

We feed an input image of "size $224 \times 224 \times 3$" to the ResNext-50 model. As mentioned by the authors in [12], the building blocks shown in Figure 3.25 are strictly equivalent, and when batch normalization and ReLU are appropriately addressed, these three building blocks give the same result when trained. The authors further noted that the building block using grouped convolutions is faster and more succinct than the other two forms. In this research, we also use the grouped convolutions building block.

4.5.2 Hyperparameter Tuning

Hyperparameter parameter tuning involves a process of adjusting specific values (or factors) which can influence the training of an algorithm. In our machine learning algorithms, we use optimization algorithms to find the most optimal result (minima). Due to the iterative nature of our optimization algorithm, we need to get the results several times in order to get the most optimal curve, and as such, we need to adjust specific values, these values are referred to as *Hyperparameters*, and tuning hyperparameters properly can help achieve a well performing model. In this research, the hyperparameters which we have tuned are discussed below:

4.5.2.1 Epochs

An Epoch is when an entire dataset has been fed to the network forward and backward ONLY ONCE. It is necessary to pass the dataset through many epochs especially in the cases where we have a limited number of training examples, to allow our network train and learn better from these training examples. Using smaller number of epochs will thus prevent convergence (that is, the network is not able to properly learn) and using unnecessarily large epochs will cause a longer training time, yet without an improvement in performance.

4.5.2.2 Batch Size

The batch size is the number of training examples present in a single batch. The batch size should however not be confused with number of batches. When training a network, it is difficult for us to pass all the dataset at once to the network for training, so we divide the entire dataset into a number of batches, where each batch contains a number of training examples, called the batch size. Choosing a batch size is important as choosing a very large batch size may lead to poor generalization by the network, and choosing a very small batch size may lead the network not converging to the global optima. Also, choosing a large batch size requires large computational resources.

4.5.2.3 Iteration

Iterations are the number of batches required to complete an epoch. An iteration, thus means passing ONE batch of training examples through the network.We will thus need x number of batches iterations to complete one epoch when training a network.

4.5.2.4 Learning Rate (LR)

LR helps to check how much to adjust the model in response to the returned estimated error each time the weights of the network are updated. An extremely small LR value can cause the network to train longer and an extremely high LR may cause the network to converge too quickly to a sub-optimal solution, thus resulting in poor model performance. The learning rate values often range between 0.0 and 1.0 and is about the most important hyperparameter when a neural network is being trained [122],and according to [123], the learning rate is the one hyperparameter to tune if you are constrained by time.

4.5.2.5 Pooling Type

Pooling is a key part of convolutional neural networks as the pooling layer help to downsample the input volume, thus reducing variance and computation complexity. There are several hyperparameters to be adjusted in the pooling layer, such as the stride value and pooling kernel size. However, another key decision to be made is the type of pooling to use. Two commonly used pooling methods are Average (Avg) Pooling and Maximum (Max) Pooling. The max pooling method works by representing a part of of an image with the highest pixel value in the region, while avg pooling computes the average (by adding all pixel values and dividing by total number of pixel values) and represent that part of the image with the average value. These two pooling technique have their advantages and disadvantages, for example, max pooling extracts the most important features such as edges, while average pooling helps to retain more information in cases where the edges of the features are localized. Using the right type of pooling also often depends on the task and type of data.

4.5.2.6 Weight Initialization

It is important to randomly initialize weights before training the network, as this helps to prevent layer activation outputs from vanishing when doing a forward pass.A vanishing or exploding result may lead to non-convergence of the network as loss gradients may be too small or large to flow backwards.

4.5.2.7 Activation Function

The activation function helps to decide whether a neuron should be activated or not. The activation function makes this decision by adding bias to the weighted sum. There are several activation functions which can be used such as Sigmoid, Tanh, Softmax and ReLU. The ReLU is the most widely used activation functions between layers, while the Softmax is the most widely used at the last layer of neural networks for classification problems.

4.5.2.8 Optimizer

Optimizers (also called optimization algorithms) help us to minimize (or maximize) Objective functions (also called Error functions). These optimizers help us to train our machine learning algorithms until we achieve at the best result (error rate). There are several optimization algorithms such as Gradient Descent [124] (one of the most popular optimization algorithms), Adagrad [125], AdaDelta [126], Adam [127] and many more.The Adam (Adaptive Moment Estimation) optimizer in practice works very well and compares favourably to other optimization algorithms because of its fast convergence and speedy learnability. The Adam optimizer conquers problems faced with other optimization algorithms such vanishing learning rate and slow convergence.

4.5.2.9 Early Stopping

Early stopping is an hyperparameter used to reduce the risk of overfitting when training neural networks. Early stopping helps to stop the training procedure of the network (even if the network is yet to train for the specified number of epochs) if the model's performance is not improving with more training.

4.5.2.10 Cost Function

The cost function helps to measure the error between the network's predicted values and the true (or expected) values. A cost function can be formed in any way depending on the task at hand, the ultimate goal is to either minimize the error or maximize the reward of a cost function. There are several cost functions which can be used in training neural networks such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Cross Entropy Loss and many others.

4.5.2.11 Dropout

According to [128], Srivasta et al. mentioned that a good way to reduce overfitting when training a neural network is to repeatedly eliminate randomly selected nodes and their corresponding connections. Dropout

regularization helps the training process noisy, thereby forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.

4.5.2.12 Batch Normalization

Batch normalization is a method that helps to standardize the input to a network. According to [129], training a neural network is complicated due to the changes in the distribution of each layer's input as the parameters of the previous layers change when training the network. This change thus makes training the network slow because we have to use a lower learning rate and careful initialization of parameters, thereby making it very difficult to train networks with saturating nonlinearities. The argument behind batch normalization is, to speed learning, we normalize the input layer by scaling and adjusting the activations, and if the input layer benefits from this approach, we can extend this benefit to other hidden layers of the network to achieve a general performance.

4.6 Metrics

Metrics are used to measure or evaluate the performance of machine learning models during training or testing. Metrics are quite different from loss/cost functions which we discussed in Section 4.4.2.10. A metric does not need to be differentiable, however, if a performance metric is differentiable, such metric can as well be used as a loss function, for example, the Mean Square Error (MSE).

4.6.1 Confusion Matrix (CM)

The Confusion Matrix is not a metric itself, but an error matrix showing the results of predictions from a trained model. The Confusion matrix shows a tabular visualization of the predicted values of a model versus the actual values. A simple Confusion Matrix shows the predictions against the ground truth for a binary classification. In Confusion Matrix, we use terms like True Positive, True Negative, False Positive and False Negative. These terms are discussed further below:

- True Positive (TP): True Positive refers to the number of predictions which are **correctly predicted** as belonging to the positive class. TP indicates that the model predicts the instances given to it as belonging to the positive class and those instances are in fact, belonging to the positive class in the ground truth.
- True Negative (TN): True Negative refers to the number of predictions which are correctly predicted as belonging to the negative class. TN indicates that the model predicts the instances given to it as belonging to the negative class and those instances are in fact, belonging to the negative class in the ground truth.
- False Positive (FP): False Positive refers to the number of predictions which are wrongly predicted as belonging to the positive class. FP indicates that the model predicts the instances given to it as belonging to the positive class, however, in fact, those instances belong to the negative class in the ground truth.

		Ground Truth	
		Positive	Negative
Predictions	Positive	TP	FP
	Negative	FN	TN

Table 4.1: A simple Confusion Matrix

		Ground Truth		
		Drowsy	Yawning	Alert
Predictions	Drowsy	DD	DY	DA
	Yawning	YD	YY	YA
	Alert	AD	AY	AA

Table 4.2: A three-class Confusion Matrix: A represents Alert, D represents Drowsy and Y represents Yawning. The first letter in each cell corresponds to the prediction, and the second letter corresponds to the ground truth, for example, DD represents samples predicted as Drowsy and are actually Drowsy.

- False Negative (FN): False Negative refers to the number of predictions which are wrongly predicted as belonging to the negative class. FN indicates that the model predicts the instances given to it as belonging to the negative class, however, in fact, those instances belong to the positive class in the ground truth.

A simple Confusion Matrix (CM) is shown in Table 4.2. For a more complex Confusion Matrix (which applies to multi-class classification problems), the CM will be of dimension $m \times m$, where m is the number of classes present in the data samples. Table 4.3 shows a Confusion Matrix for a three-class classification problem.

In Table 4.3, the diagonal cells from top left to bottom right show the correct predictions. All other cells, show the misclassification of the samples and into what class.

4.6.2 Accuracy

The most common and simplest form of metrics is the "classification accuracy", when it comes to classification problems. The classification accuracy is the total number of correct predictions divided by the total number of predictions, multiplied by 100. Mathematically, the classification accuracy can be represented as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4.1}$$

In our study, Sometimes, the accuracy of the model does not give us full knowledge of a model's performance, for example, in cases where there is class imbalance in the data samples used to evaluate the model's performance. For example, in a scenario where there is a test data comprises of 90% of class A and 10% of class B, even if the model predicts all of class A correctly and all of class B incorrectly, the accuracy of the model tricks us into thinking the model's performance is 90%, whereas the model actually learned nothing. It is thus not sufficient sometimes to only use the accuracy to evaluate the model's performance.

4.6.3 Precision

The precision is a metric used to measure the fraction of positive samples which have been predicted correctly to all the samples which have been predicted as positive. The precision is computed mathematically as:

$$\text{Precision} = \frac{TP}{TP + TN}. \tag{4.2}$$

The precision aims to check what samples are actually positive of all the samples predicted as positive by the model.

4.6.4 Recall

The recall is another metric which is similar to precision, however, recall is used to measure the fraction correctly predicted samples to all ground truth positive samples. The recall is calculated mathematically as:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{4.3}$$

The recall aims to check of all the samples which are actually positive, how many did the model correctly predict as positive. Sometimes, it becomes difficult to choose a better metric between the recall and the precision, especially when one is much higher than the other, and, many often times, we find having to increase one at the expense of the other. To solve this problem of having to choose either of these metrics, we use another metric called F1 Score

4.6.5 F1 Score

The F1 score is the harmonic mean of precision and recall, we use the harmonic mean so as to allow for penalizing the extreme values. The F1 score is calculated mathematically as:

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{4.4}$$

that is;

$$\text{F1 Score} = 2 * \frac{\frac{TP}{TP+TN} * \frac{TP}{TP+FN}}{\frac{TP}{TP+TN} + \frac{TP}{TP+FN}}. \tag{4.5}$$

4.6.6 Speed

In our research, we have included speed as a metric to evaluate our model's performance, this is because, in developing an effective real-time monitoring system, the correct predictions is not the only necessary factor.

The speed of processing and prediction is highly necessary, as feedback are required in shortest time possible. The speed is mathematically calculated as:

$$\text{Speed} = \frac{\text{Time taken to predict all samples}}{\text{Total number of predictions}}. \tag{4.6}$$

4.6.7 Video Error (VE) and Average Video Error (AVE)

The "Video Error" is the main metric used to evaluate our overall model. The video error is the percentage error between the actual current alertness value (CAV) at the end of a video and the current alertness value ($C\hat{A}V$) produced from our network. The video error is mathematically calculated as:

$$VE = \frac{|CAV - C\hat{A}V|}{100};\ 0 \leq CAV \leq 100;\ 0 \leq CAV \leq 100, \tag{4.7}$$

where VE is the video error, actual CAV is the alertness value at the end of the video and $C\hat{A}V$ is the alertness value produced from the network. The CAV and $C\hat{A}V$ minimum and maximum value are 0 and 100 respectively. The best VE is almost 0, when $C\hat{A}V \approx CAV$, thus, our aim is to get a VE value closest as possible to 0. The VE is majorly used to evaluate the model performance on one video, however, given N test videos, the "Average Video Error (AVE)" is obtained by getting the sum of all video errors divided by the number of test videos N. The AVE is mathematically calculated as:

$$AVE = \frac{1}{N}\sum_{i=1}^{N} VE_i, \tag{4.8}$$

where $i = 1...N$, and VE_i is the ith video error.

In this research, we use only three of the metrics discussed above which are Accuracy (obtained from frames classification), Video Error/Average Video Error (obtained from overall video) and Speed. We will also be using the confusion matrix to evaluate the performance of the model and its misclassifications.

4.7 System Details

We used Python to build all the models considered in this research. We chose Python largely because it is easy to learn, and also it has a wide range of libraries which have community support. The general libraries used include Numpy (for basic scientific computing), Google-Images-Search and BeautifulSoup (for image scraping), Matplotlib and Pyplot (for visualization), OpenCV and PIL (for image processing and manipulation) and Flask (for the web API). The deep learning libraries include Tensorflow (as back end) and Keras (for the front end). We also used manual image editors such as: Windows Photos and Linux GIMP.

The hardware specifications of the system we used are: **RAM**: 16 GB, **GPU**: 8 GB, and **CPU**: 8 cores (Intel)

4.8 Conclusion

In this Chapter, we presented the methodology for driver alertness monitoring. Section 4.2 discusses the overall framework, starting from the input layer to the final output layer. In Section 4.3, discussed how we have collected the dataset which we used in this study, while in Section 4.4, we discussed how we preprocessed the dataset collected. We combined images from different databases with some images we compiled through the Bing and Google APIs. We further discussed the different face detection algorithms readily available to us and made comparisons, thereby, selecting the two best suited algorithms for our study which are Dlib and MTCNN. The Dlib is the default face detector and only passes an image to the MTCNN if it is unable to detect a face. We also discussed data augmentation techniques as a way to increase variations in our dataset. In Section 4.5, we discussed the CNN variants we used in this study and also discussed the effects of hyperparameter tuning on a model's performance. In Section 4.6, we discussed several metrics that could be used to measure our model, but we eventually selected only three metrics which are Accuracy, Speed and Video Error/Average Video Error. We also selected the confusion matrix to help us better visualize the performance of the model on the different classes. Lastly, Section 4.7 discusses the details of the software and hardware used in this study. Based on this Chapter, our results and findings are presented in Chapter 5.

Chapter 5

Results and Discussions

5.1 Introduction

In the previous Chapter, we discussed the methodology for this research. We discussed also hyperparameters tuning and architectures (LeNet-5, AlexNet, VGG-16, Inception-ResNet-V2 and ResNext-50) used in this study for driver alertness monitoring. To monitor our model performance, we used K-1 fold validation. The K-1 fold validation is used due to the long time for which some of our architectures train. In this Chapter, we discuss our findings and results from running experiments on the data and architectures discussed in Chapter 4.

5.2 LeNet-5

The LeNet algorithm was trained on a 3-classification task: Normal, Drowsy and Yawning. The input face images which are 9400 in total were also resized to $32 \times 32 \times 1$ according to the LeNet-5 architecture. We also tuned several hyperparameters, such as pooling type and learning rate. We also investigated the effect of batch normalization [129] which was introduced several years after the LeNet-5 was introduced by Yann Lecun. We also did data augmentation in order to make our model robust to several varying conditions. The LeNet-5 was trained with 6016 and 1504 samples were used for validation while 1880 (containing 735 Normal, 400 yawning and 745 drowsy) samples were used to evaluate the network's performance. In this section, we discuss the performances of LeNet with different hyperparameters and operations.

5.2.1 LeNet-5 without Batch Normalization, Pooling type: Maxpooling

The LeNet-5 performed well on our dataset without the batch normalization. Using the maxpooling and a learning rate of 0.001, the network trained for 5 minutes and 38 seconds for 25 epochs, achieving a 97.34% accuracy on the training data and 97.04% accuracy on validation data. When the learning rate is increased to 0.01, the network converged faster and achieved the same accuracy of 97.34% accuracy on training data and 97% accuracy on validation data only after 9 epochs and after 25 epochs, the network achieved a 99.37% on training data and 97.84% accuracy on the validation data. On the test set, the network with 0.001 learning rate achieved an accuracy of 85.6% on the test data while the network trained with learning rate of 0.01 achieved an accuracy of 85.7% accuracy on the test data.

Hyper-parameters	LR = 0.001, Pooling = Maxpool and BN = No	LR = 0.01, Pooling = Maxpool and BN = No
Metrics	Test Accuracy	Test Accuracy
Values	85.6	85.7

Table 5.1: Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using maximum pooling. LR denotes Learning Rate, BN denotes Batch Normalization.

Hyper-parameters	LR = 0.001, Pooling = Avgpool and BN = No	LR = 0.01, Pooling = Avgpool and BN = No
Metrics	Test Accuracy	Test Accuracy
Values	85.8	86.2

Table 5.2: Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using maximum pooling. LR denotes Learning Rate, BN denotes Batch Normalization.

The diagrams in Figure 5.1 shows the performances of LeNet-5 using maxpooling without batch normalization at learning rates of 0.001 and 0.01 and in Figure 5.2, we show the confusion matrix of predictions when the model was used fed 1880 test samples.

Table 5.1 shows the results of the LeNet-5 when trained without batch normalization and using maximum pooling.

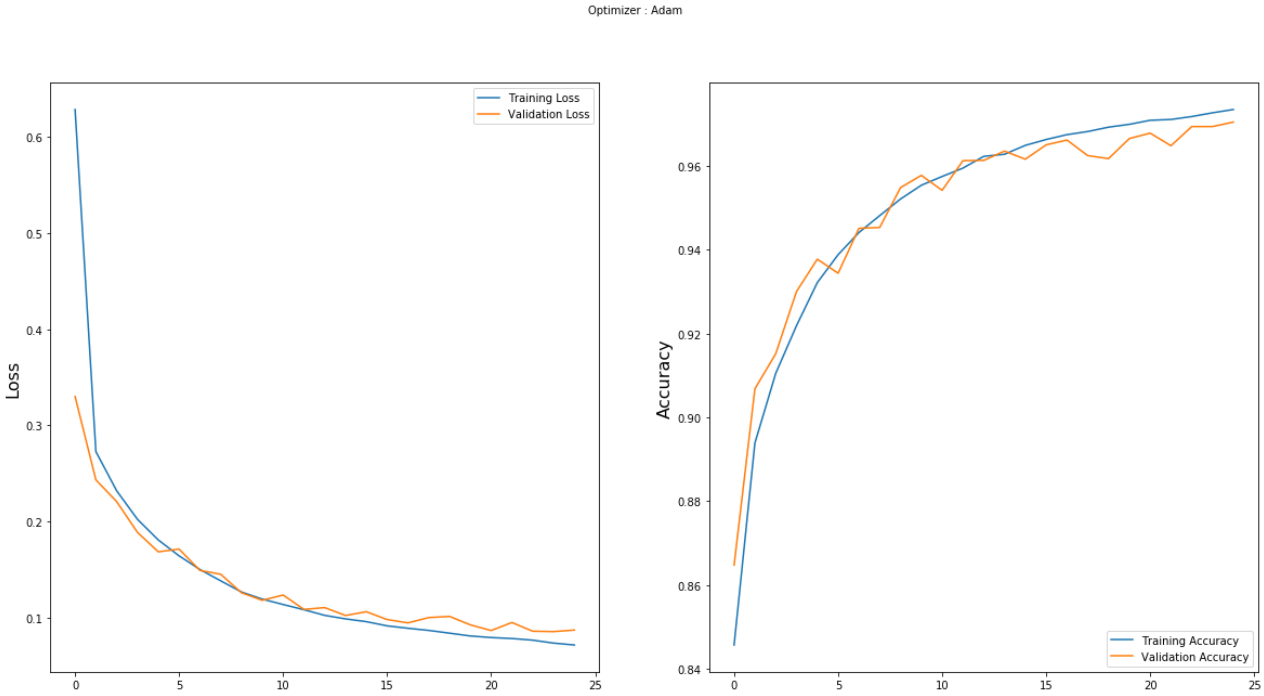
5.2.2 LeNet-5 without Batch Normalization, Pooling type: Average Pooling (Avgpooling)

We further investigated the performance of LeNet-5 when the average pooling type is used, at the same learning rates without batch normalization. For a learning rate of 0.001, the model trained for 5 minutes and 12 seconds, achieving a training accuracy of 97.48% , and a validation accuracy of 97.11%. When the learning rate is adjusted to 0.01, the model trained for 5 minutes and 3 seconds (10 seconds less than learning rate of 0.001), achieving a training accuracy of 99.05% and validation accuracy of 98.34%. On the testing samples, the model with learning rate of 0.001 achieved an 85.8% accuracy while the model with learning rate of 0.01 achieved an accuracy of 86.2%.

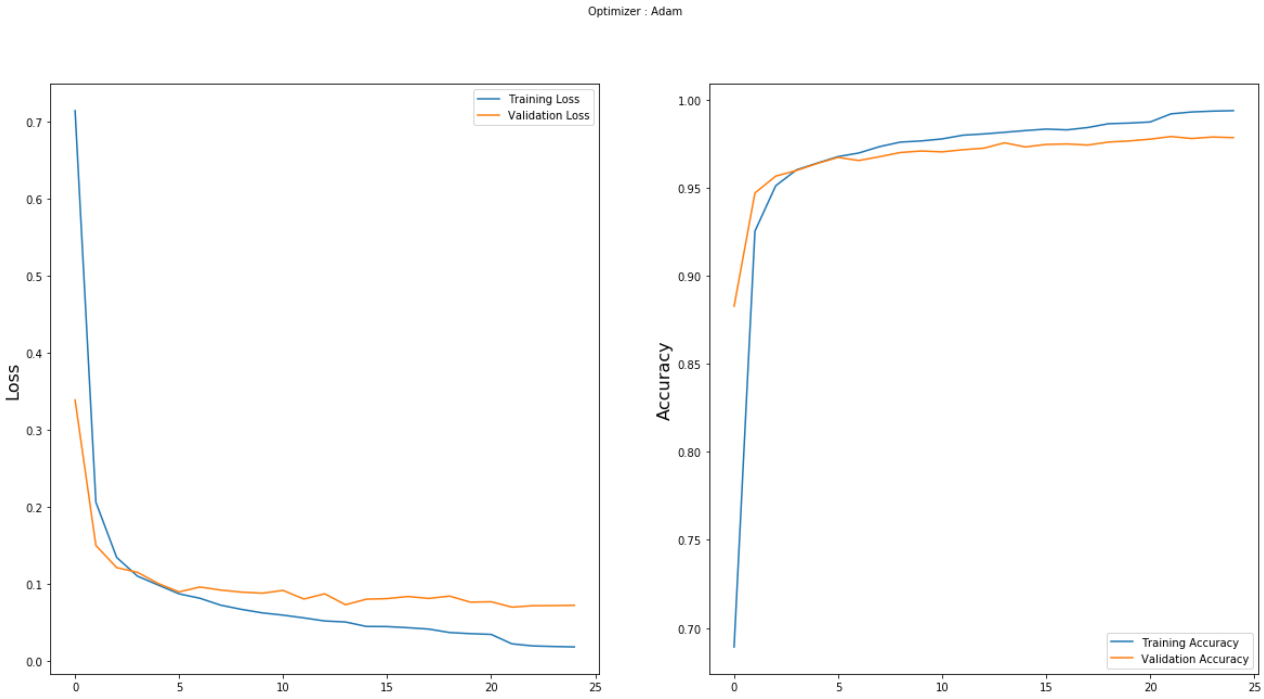
The diagrams in Figure 5.3 show the performances of LeNet-5 using avgpooling without batch normalization at learning rates of 0.001 and 0.01 while the diagrams in Figure 5.4 show the confusion matrix when the model was used to predict 1880 samples. Table 5.2 shows the results of the LeNet-5 when trained without batch normalization and using average pooling.

5.2.3 LeNet-5 with Batch Normalization before Activation

As mentioned in Section 4.4.2.12, in literature, batch normalization is said to improve performance of deep learning models. We thus, included batch normalization after each convolutional layer and 2 fully connected

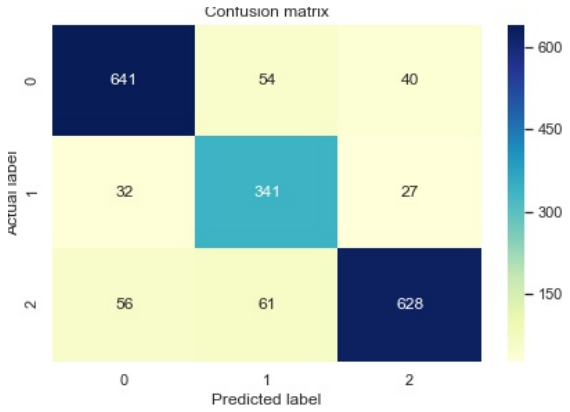


(a) LeNet-5: LR=0.001, Pooling=Maxpooling, Batch-Normalization=No

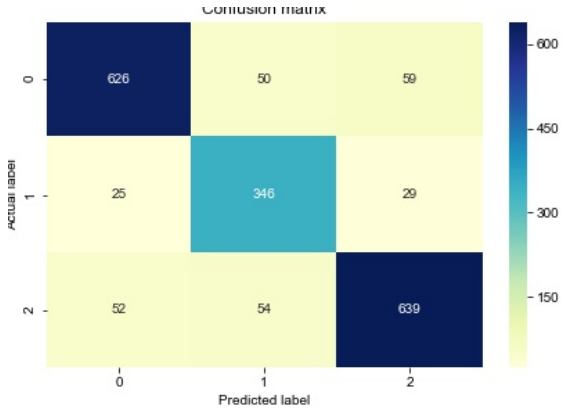


(b) LeNet-5: LR=0.01, Pooling=Maxpooling, Batch-Normalization=No

Figure 5.1: Performance plot of LeNet-5 using maxpooling without batch normalization during training



(a) LeNet-5: LR=0.001, Pooling=Maxpooling, Batch-Normalization=No



(b) LeNet-5: LR=0.01, Pooling=Maxpooling, Batch-Normalization=No

Figure 5.2: Confusion Matrix of LeNet model trained without Batch Normalization using Maxpooling

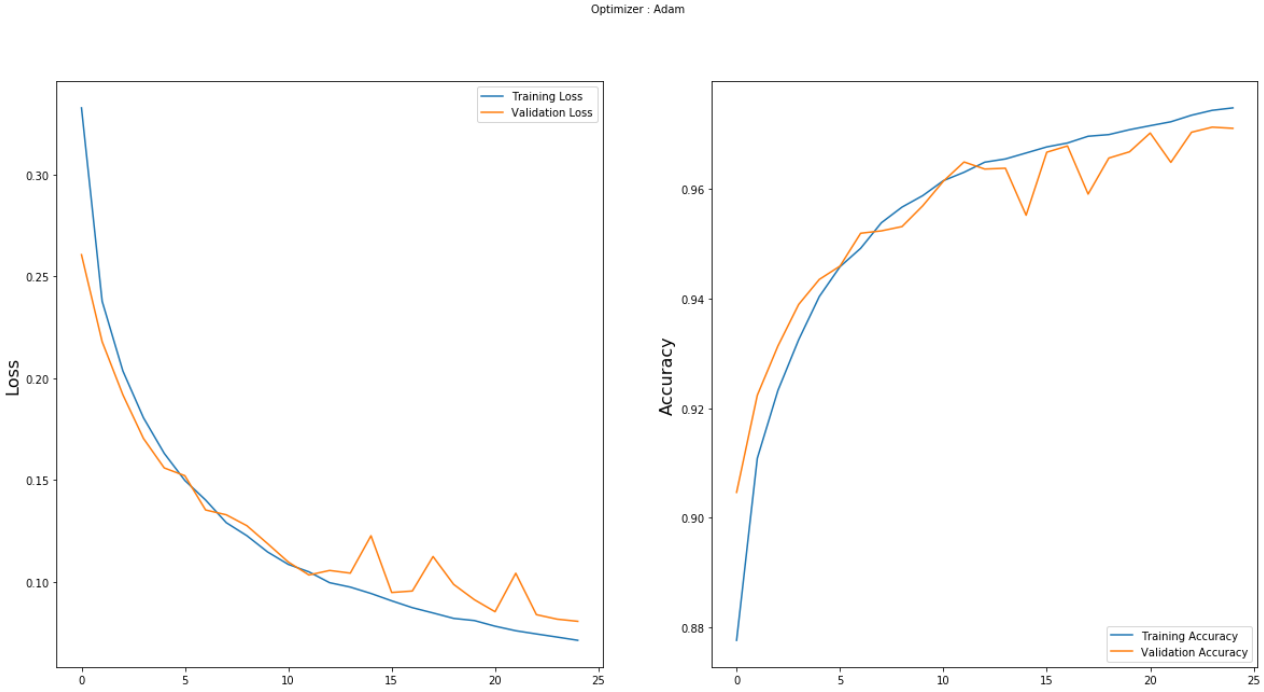
Hyper-parameters	LR = 0.001, Pooling = Avgpool and BN = Before activation	LR = 0.01, Pooling = Avgpool and BN = Before activation
Metrics	Test Accuracy	Test Accuracy
Values	87.29	86.5

Table 5.3: Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using average pooling and batch normalization. LR denotes Learning Rate, BN denotes Batch Normalization.

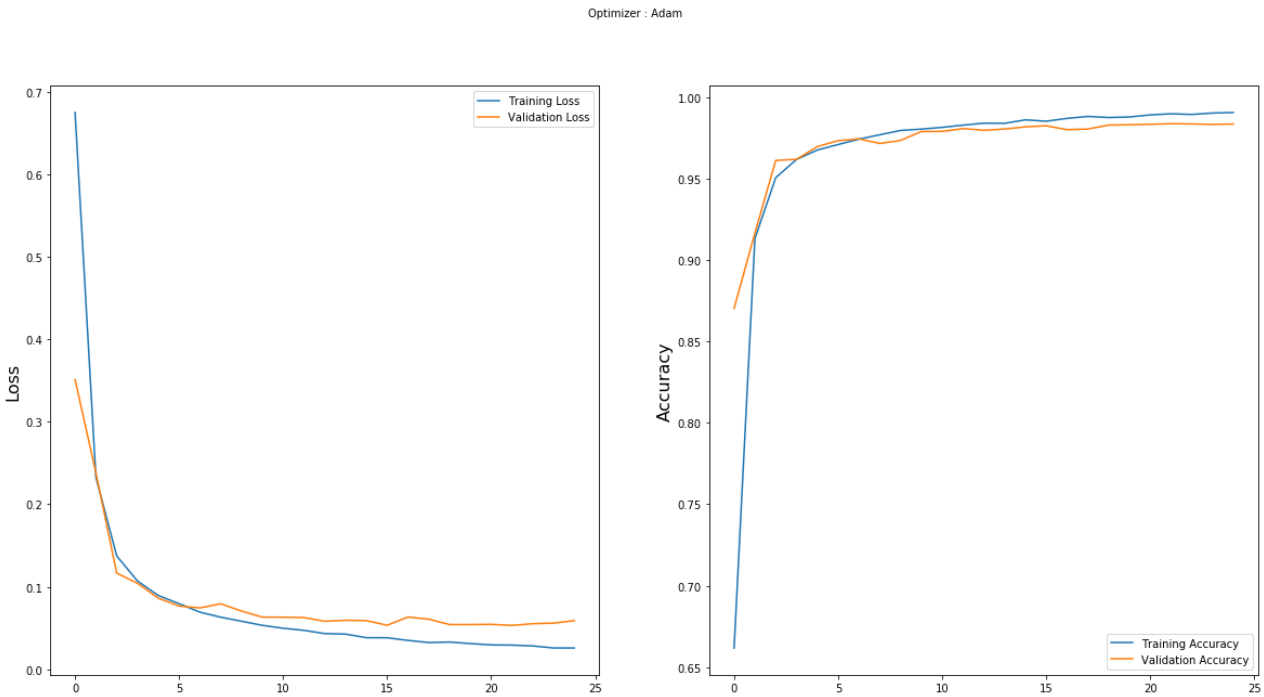
layers to observe how batch normalization affects the training of a network. From our experiments, we observed that, different results are obtained from where batch normalization is used. In Figure 5.5, we show the results obtained from using batch normalization before the activation function. We also chose the average pooling type since it gives the best performance when comparing the results from sections 5.2.1.1 and 5.2.1.2. Experimental results show that using batch normalization before activation function on our dataset at a learning rate of 0.001 yields a training accuracy of 97% and validation accuracy of 97.16% after 25 epochs. The network had peaked at 97% validation accuracy from the 20th epoch and maintained this accuracy while steadily increasing the training accuracy. When evaluated on the testing data, the network correctly predicted 87.29% of the data. When trained at a learning rate of 0.01, the network had peaked at 97.5% accuracy on the training data and 97.04% on the validation data only after the 4th epoch, the network improved to about 98% accuracy on training and validation data from the 7th epoch and maintained this accuracy with few improvements, the network seemed to start overfitting the data from 19th epoch, thus we reduced the learning rate to 0.001. After 25 epochs, the network produced a training accuracy of 99% and validation accuracy of 98.79%. However, when evaluated on the test set, the network only achieved 86.5% accuracy which is a large drop in testing accuracy when trained for about half the number of epochs (where testing accuracy was 89%).

The diagrams in figures 5.5 show the performances of LeNet-5 using average pooling with batch normalization before the activation function at learning rates of 0.001 and 0.01 while the diagrams in Figure 5.6 show the confusion matrix when the model was used to predict 1880 samples.

Table 5.3 shows the results of the LeNet-5 when trained with batch normalization before the activation function and using average pooling.

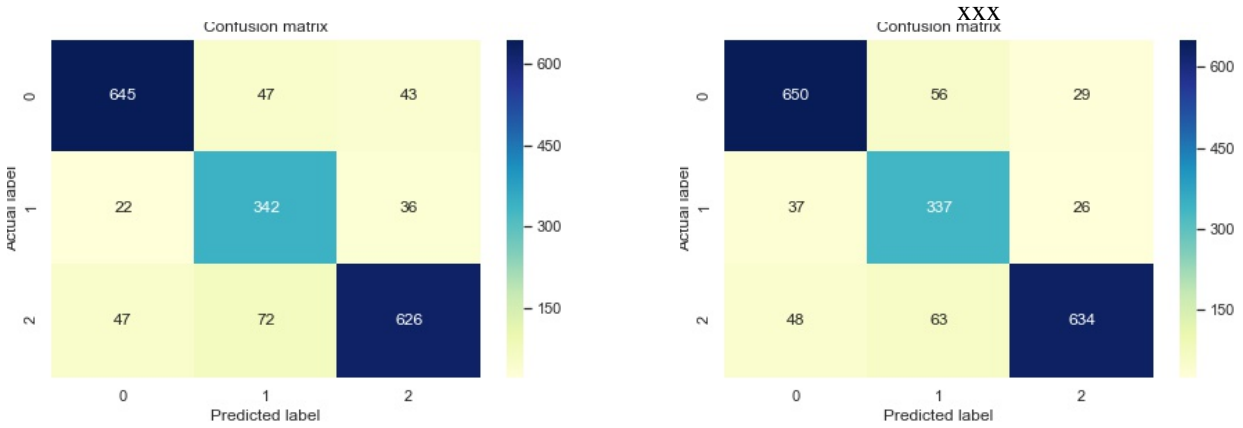


(a) LeNet-5: LR=0.001, Pooling=Avgpooling, Batch-Normalization=No



(b) LeNet-5: LR=0.01, Pooling=Avgpooling, Batch-Normalization=No

Figure 5.3: Performance plot of LeNet-5 without batch normalization during training



(a) LeNet-5: LR=0.001, Pooling=Avgpooling, Batch-Normalization=No

(b) LeNet-5: LR=0.01, Pooling=Avgpooling, Batch-Normalization=No

Figure 5.4: Confusion Matrix of LeNet model trained without Batch Normalization using Avgpooling

Hyper-parameters	LR = 0.001, Pooling = Avgpool and BN = After activation	LR = 0.01, Pooling = Avgpool and BN = After activation
Metrics	Test Accuracy	Test Accuracy
Values	82.7	86.76

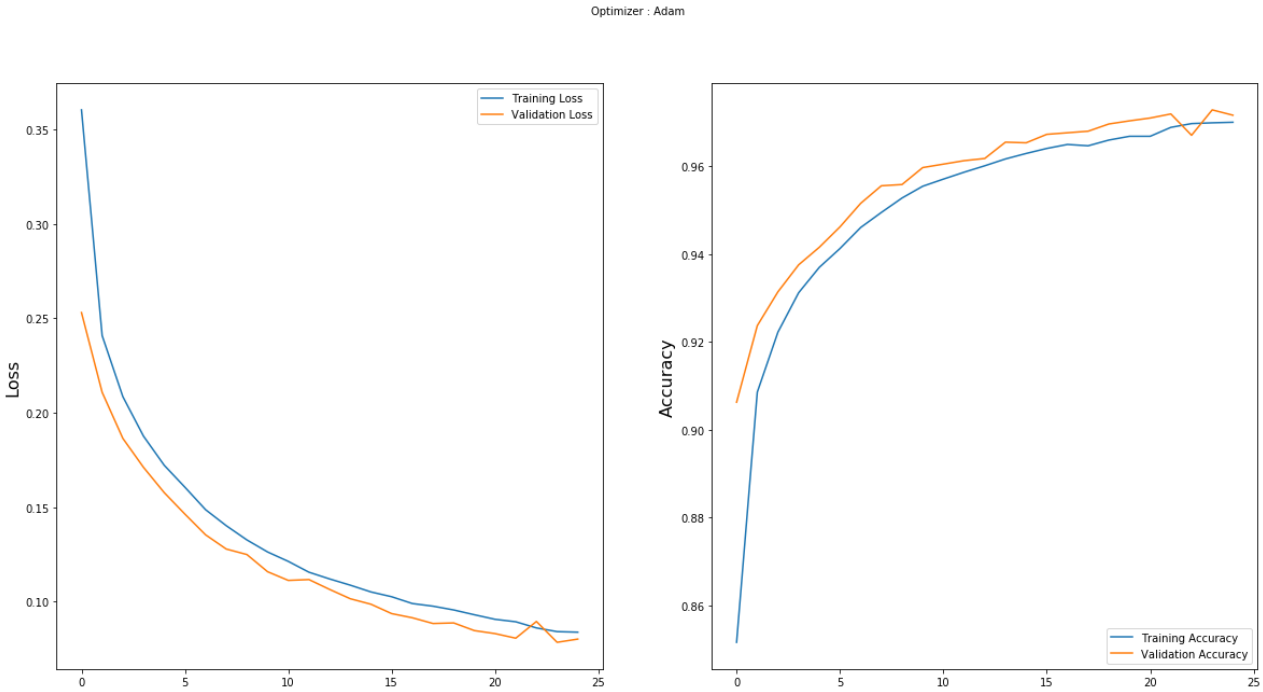
Table 5.4: Results comparing model performance of LeNet-5 at learning rates of 0.001 and 0.01 using average pooling and batch normalization. LR denotes Learning Rate, BN denotes Batch Normalization.

5.2.4 LeNet-5 with Batch Normalization after Activation

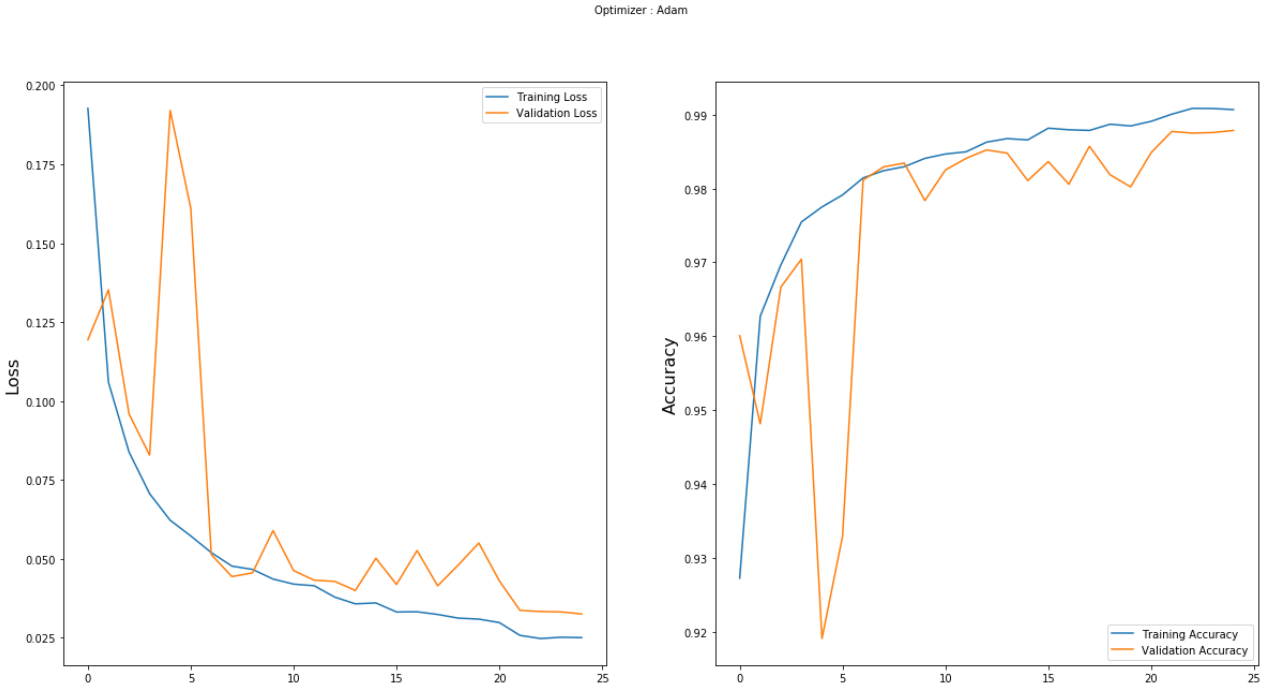
We further investigate the performance of the network when the batch normalization is used after the activation function. Results show that the network at a learning rate of 0.001 achieved 97.59% accuracy on the training data and 97.52% accuracy on the validation data after 25 epochs. When evaluated on the test set, the network achieved only 82.7% accuracy. We further increase the learning rate to 0.01, and the network achieved 98% accuracy on the training and validation data. Although, as expected, the network had peaked at 97% accuracy only after the 4th epoch and had peaked at 98% training accuracy after the 8th epoch and 98% validation accuracy after the 11th epoch. The network maintained this accuracy in a little range, however, the network started to overfit at the 24th epoch, thus, we reduced the learning rate to 0.001. The network after 25 epochs eventually dropped in validation accuracy to 96% while the training accuracy was maintained at 98%, this tells us that the model overiftted badly and would have performed excellently if trained for half the number epochs. The testing set was used to evaluate the model after 25 epochs and yielded an accuracy of only 86.76%.

The diagrams in Figure 5.7 show the performance plot of the network when batch normalization is used after activation at learning rates of 0.001 and 0.01 over 25 epochs while the diagrams in Figure 5.8 show the confusion matrix when the model was used to predict 1880 samples.

Table 5.4 shows the results of the LeNet-5 when trained with batch normalization after the activation function and using average pooling.

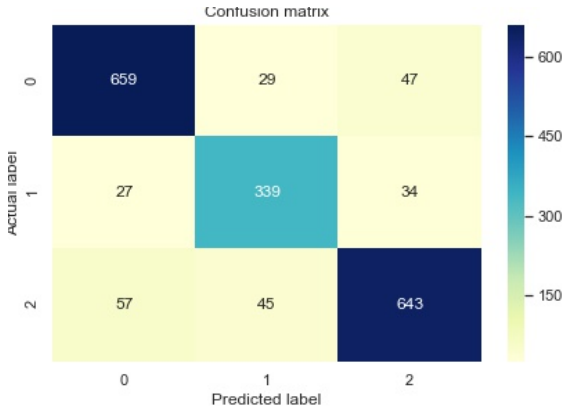


(a) LeNet-5: LR=0.001, Pooling=AvgPooling, Batch-Normalization=Before Activation

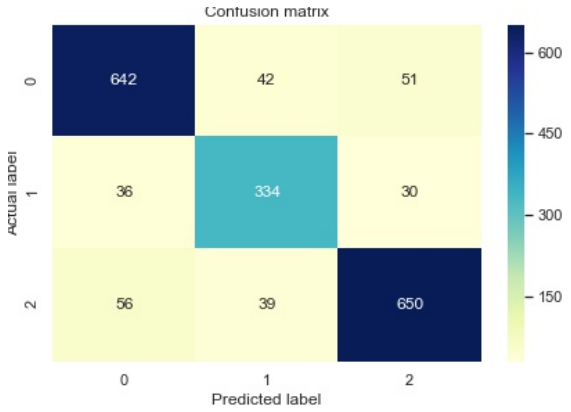


(b) LeNet-5: LR=0.01, Pooling=AvgPooling, Batch-Normalization=Before Activation

Figure 5.5: Performance plot of LeNet-5 with batch normalization before activation at different learning rates.



(a) LeNet-5: LR=0.001, Pooling=Avgpooling, Batch-Normalization=Before Activation



(b) LeNet-5: LR=0.01, Pooling=Avgpooling, Batch-Normalization=Before Activation

Figure 5.6: Confusion Matrix of LeNet model trained with Batch Normalization before activation using Avgpooling

Hyperparameters	LR = 0.001, Pooling = Avgpool and BN = After activation
Metrics	Test Accuracy
Values	88.2

Table 5.5: Results showing AlexNet model performance at learning rate of 0.001 using average pooling and batch normalization. LR denotes Learning Rate and BN denotes Batch Normalization.

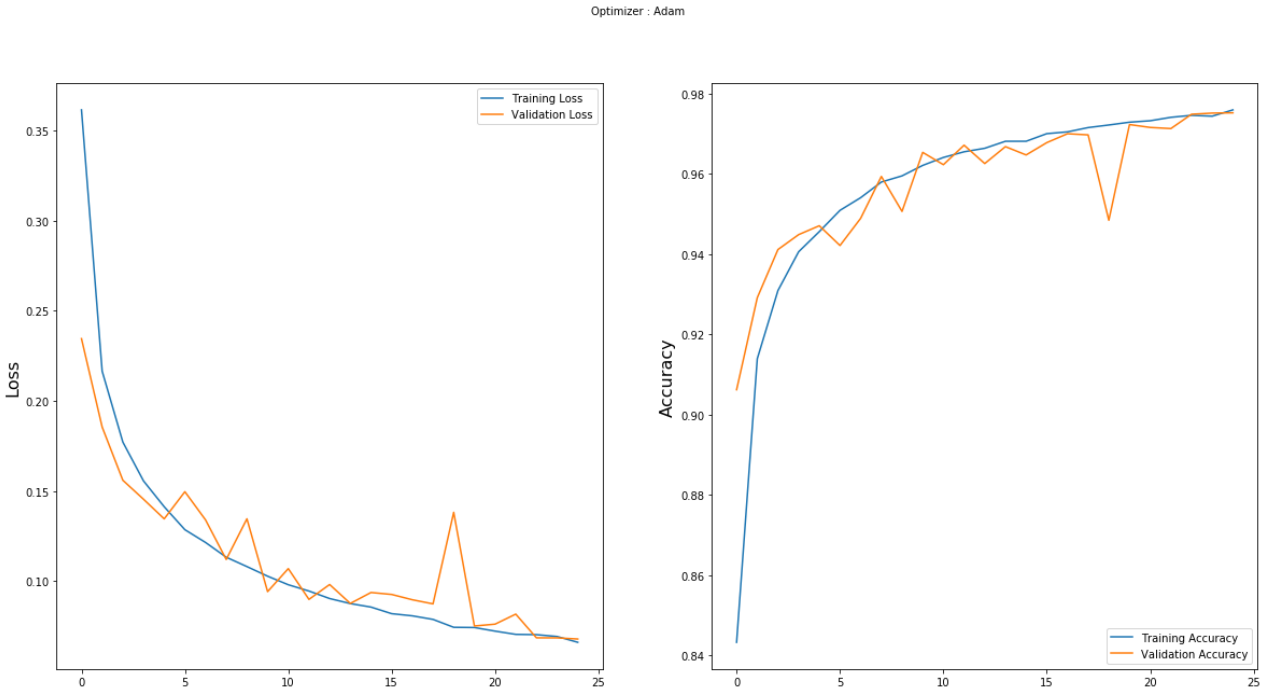
5.3 AlexNet

We fed a total of 9400 samples containing faces (2492 yawning, 4078 normal and 2830 drowsy) . These images are of dimensions $256 \times 256 \times 3$, and are divided into three classes: Normal, Yawning and Drowsy. We also did data augmentation in order to make our model robust to several varying conditions. The AlexNet was trained with 6016 for 100 epochs and 1504 samples were used for validation while 1880 (containing 735 Normal, 400 yawning and 745 drowsy) samples were used to evaluate the network's performance. Although, batch normalization was introduced after AlexNet, we have included batch normalization after each convolutional layer and fully connected layer, except for the output layer. The training performance is shown in Figure 5.9, with the training accuracy of 95.12% and validation accuracy of 97.31%. The network also achieved a result of 88.2% accuracy on the test samples.

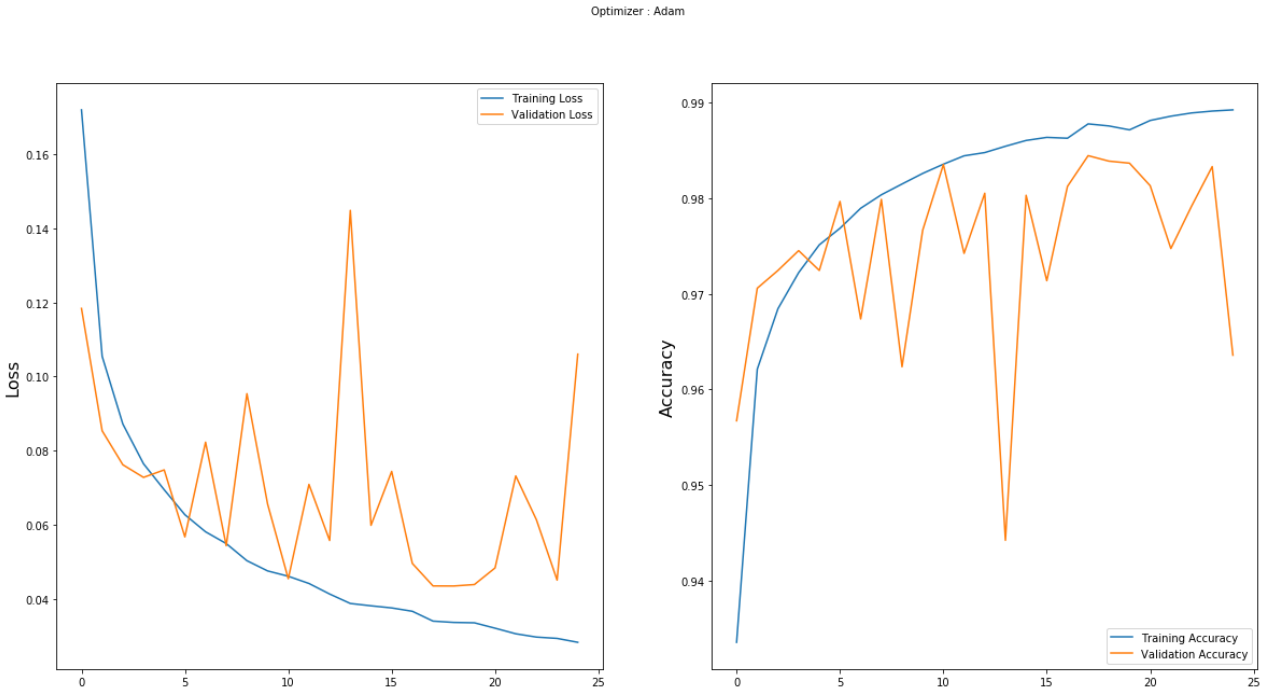
Table 5.5 shows the results of the AlexNet when tested with 1880 samples and Figure 5.10 shows the confusion matrix when the model was used to predict 1880 samples.

5.4 VGG-16

We also trained the VGG network on the same dataset which AlexNet was trained with. The dataset which contains 9400 samples, were split into training, validation and test samples. 6016 samples were used to train

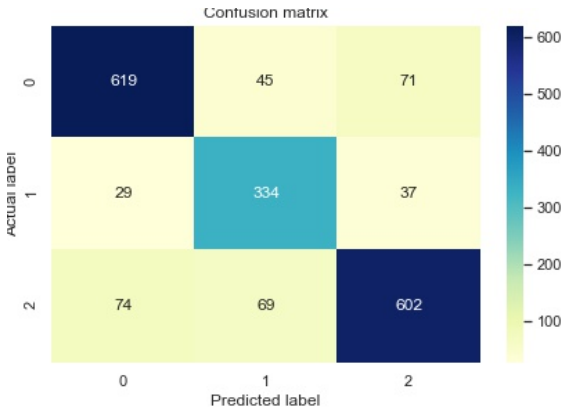


(a) LeNet-5: LR = 0.001, Pooling = AvgPooling, Batch-Normalization = Before Activation

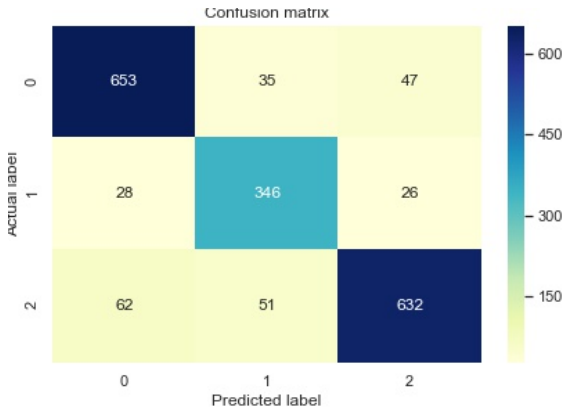


(b) LeNet-5: LR = 0.01, Pooling = AvgPooling, Batch-Normalization = After Activation

Figure 5.7: Performance plot of LeNet-5 with batch normalization after activation at different learning rates.



(a) LeNet-5: LR=0.001, Pooling=Avgpooling, Batch-Normalization=After Activation



(b) LeNet-5: LR=0.01, Pooling=Avgpooling, Batch-Normalization=After Activation

Figure 5.8: Confusion Matrix of LeNet model trained with Batch Normalization after activation using Avgpooling

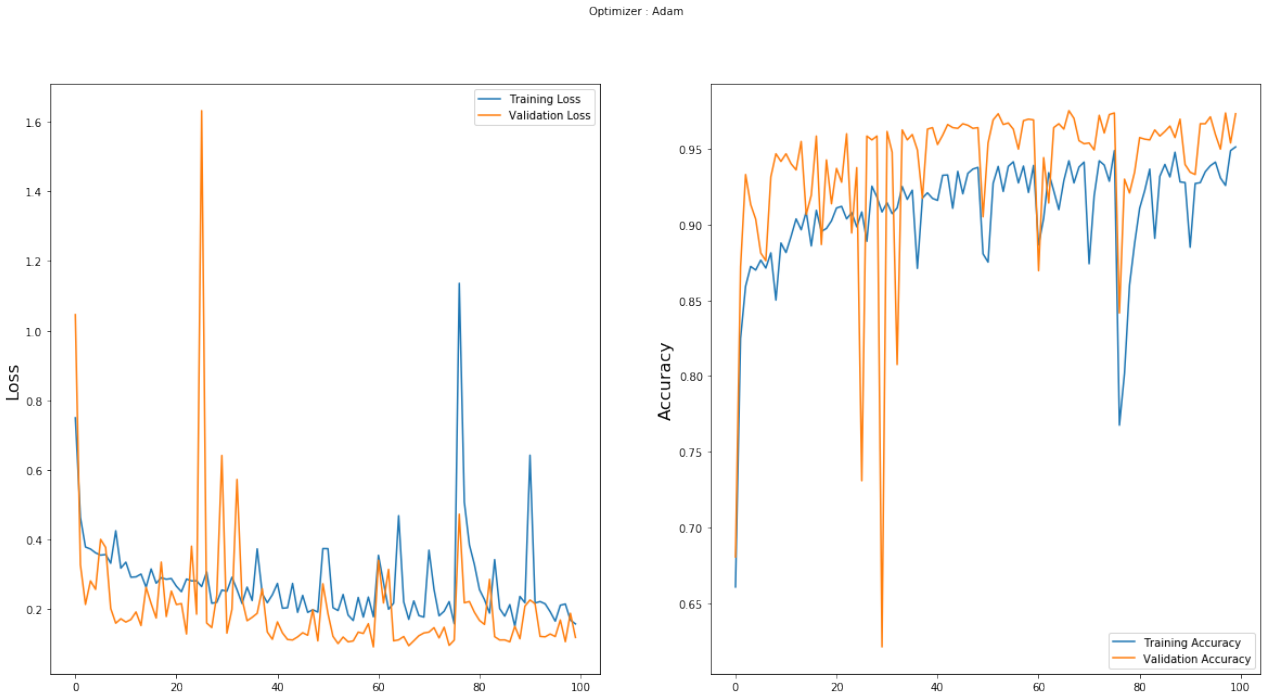


Figure 5.9: AlexNet performance plot during training

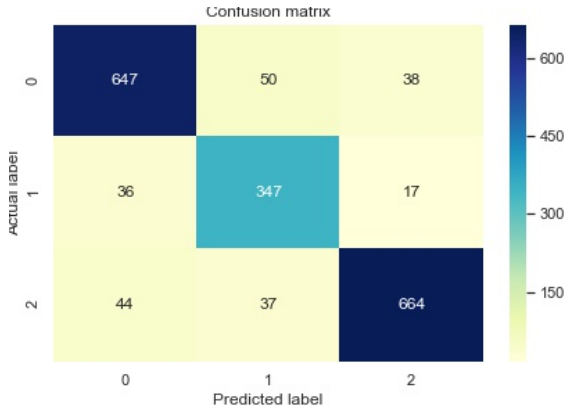


Figure 5.10: Confusion Matrix of AlexNet model trained with batch normalization

Hyperparameters	LR = 0.001, Pooling = Avgpool and BN = After activation
Metrics	Test Accuracy
Values	91.0

Table 5.6: Results showing model performance of VGG-16 at learning rate of 0.001 using average pooling and batch normalization. LR denotes Learning Rate and BN denotes Batch Normalization.

the network and 1504 samples used for validation. The network trained on the 3-class dataset achieving a training accuracy of 90.32% and validation accuracy of 92.4% after 100 epochs. When evaluated on 1880 test samples, the model achieved an accuracy of 91%. The performance plot for VGG-16 is shown in Figure 5.11.

Table 5.6 shows the results of the VGG-16 when tested with 1880 samples while the confusion matrix is shown in Figure 5.12.

5.5 Inception-ResNet V2

We also trained the inception-resnet v2 on the face dataset and it performed excellently on the data, yielding a training accuracy of 98.74% and validation accuracy of 98.32%. The network produced an accuracy of 97.2% accuracy on the test samples. The performance of the inception-resnet during training over 100 epochs is shown in Figure 5.13. Unlike the LeNet-5, AlexNet and VGGNet, we did not tune any hyperparameters other than the learning rate.

Table 5.7 shows the results of the Inception-ResNet V2 when tested with 1880 samples while Figure 5.14 shows the confusion matrix.

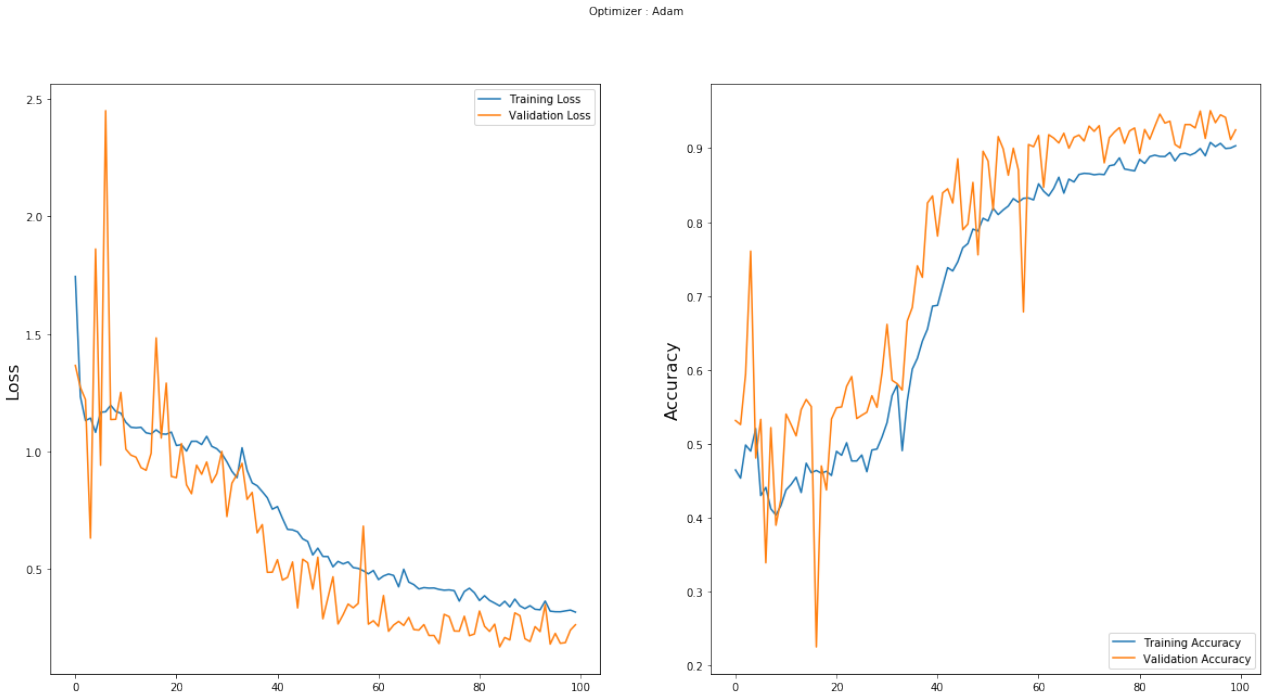


Figure 5.11: VGG-16 performance plot during training

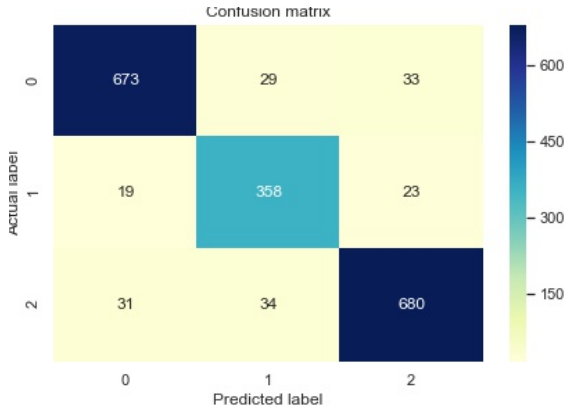


Figure 5.12: Confusion Matrix of VGG-16 model

Hyperparameter	LR = 0.001
Metrics	Test Accuracy
Values	97.2

Table 5.7: Results showing model performance of Inception ResNet V2 at learning rate of 0.001.

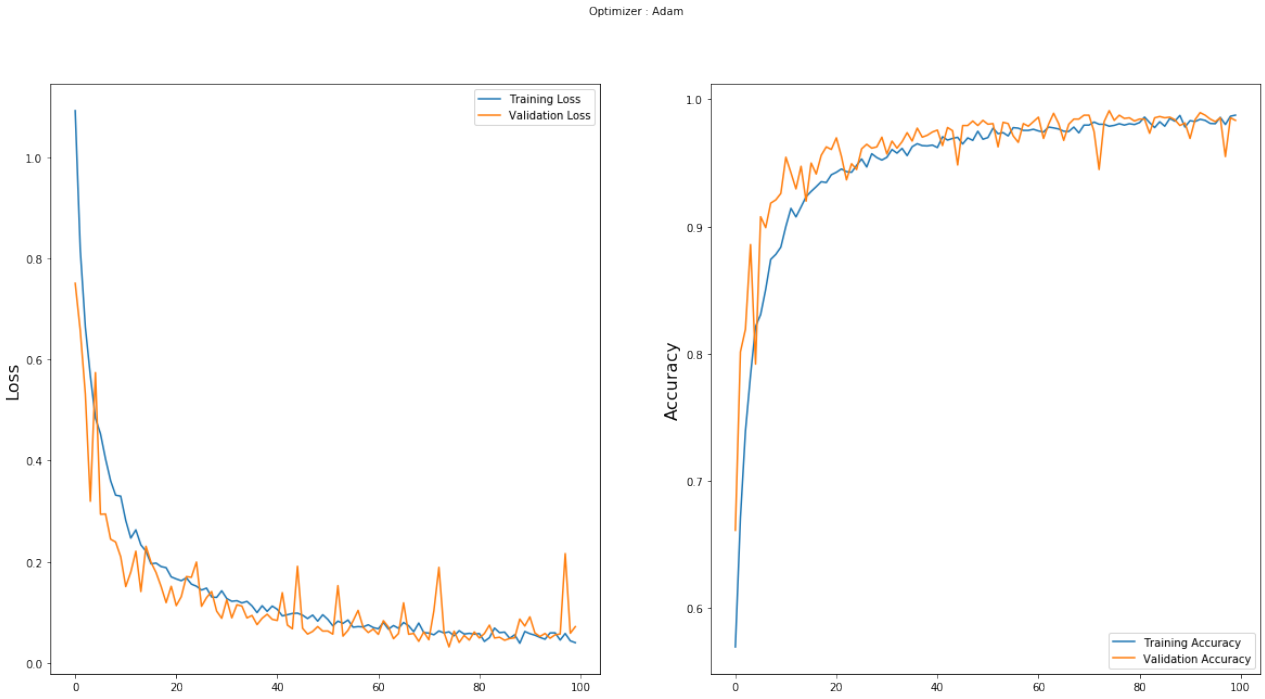


Figure 5.13: Inception ResNet V2 performance plot during training

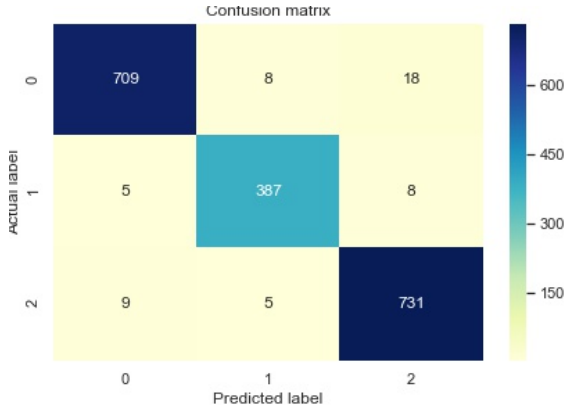


Figure 5.14: Confusion Matrix of Inception-ResNet V2 model

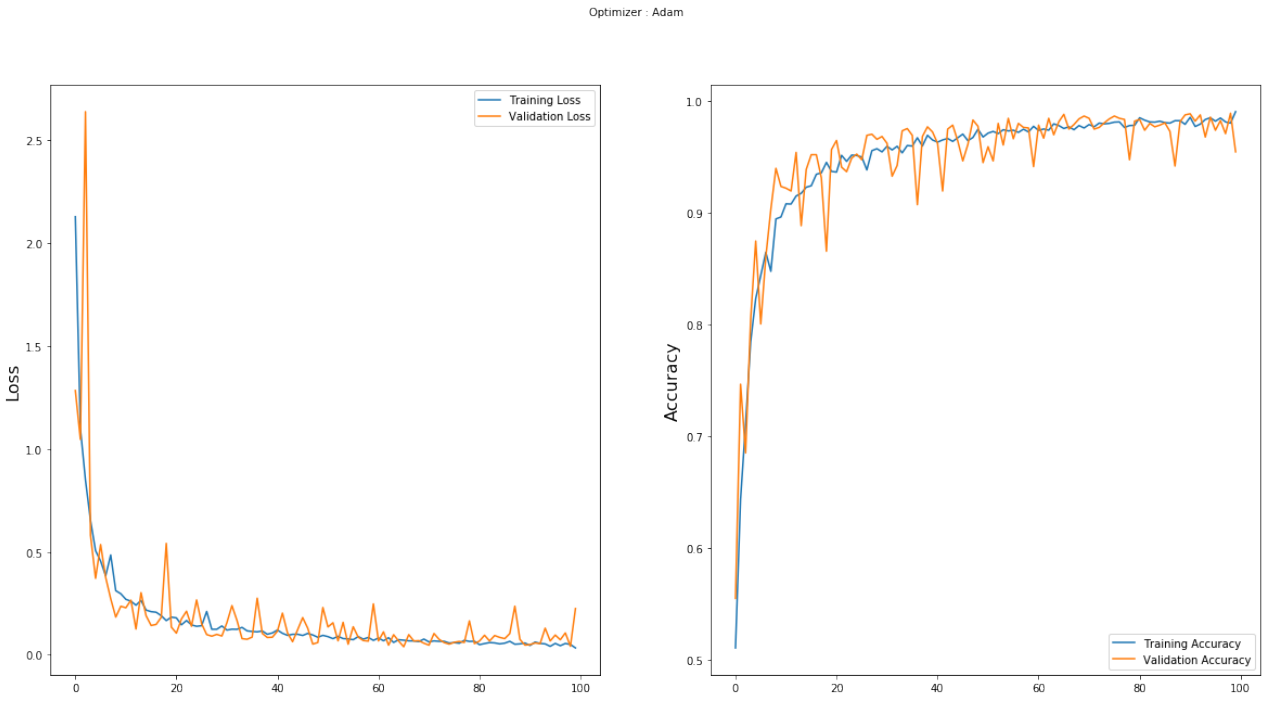


Figure 5.15: ResNext-50 performance plot during training

Hyperparameter	LR = 0.001
Metrics	Test Accuracy
Values	96.0

Table 5.8: Results showing model performance of ResNext at learning rate of 0.001.

5.6 ResNext-50

The ResNext-50 architecture also trained well on the face data and produced an accuracy of 99% on the training data and 95.4% accuracy on the validation data. On the testing set, the network performed well and predicted 96% of the test samples accurately. The performance plot of the ResNext-50 model is shown in Figure 5.15. Similar to our Inception ResNet V2, we did not make any hyperparameter tuning to this model.

Table 5.8 shows the results of the ResNext-50 when tested with 1880 samples while Figure 5.16 shows the confusion matrix.

5.7 Observations

In section 5.2, we observed that the pooling type used has a significant effect on a network's performance. We also observed, that, batch normalization has a significant effect on the network as it improved the performance

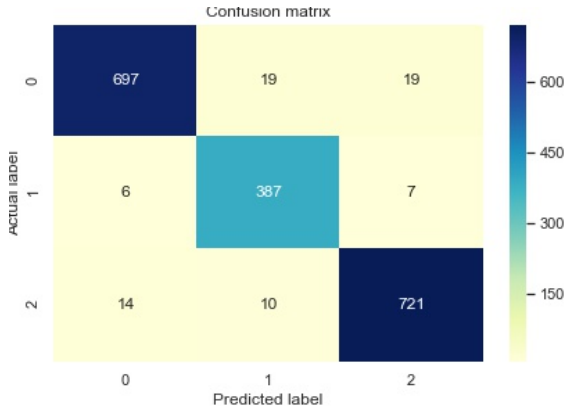


Figure 5.16: Confusion Matrix of ResNext-50 model

of the LeNet-5 compared to when batch normalization was not used. Experimental results also showed that it is best to use the batch normalization before the activation functions.

5.8 Drowsiness Level Evaluator

The drowsiness level evaluator is the final stage of our driver alertness monitoring. The evaluator is designed with specific conditions and thresholds based on our findings from previous literature. The evaluator makes use of the current alertness value (CAV) to decide the level of drowsiness a driver is currently at. A CAV of at least 80% classifies the driver as being "Alert", while a CAV of between 65% and 80% classifies the driver as being "Slightly Drowsy". If the CAV is between 50% and 65%, then, the driver is classified as being "Moderately Drowsy", but, if the CAV is between 25% and 50%, then the driver is said to be "Very Drowsy". Any CAV less than 25% classifies the driver as being dangerously drowsy. The CAV is always initially set to 100% and this value is penalized subsequently based on the facial information returned by the driver. These penalties are based on specific conditions which we gathered from literature, and these conditions are as follows:

1. According to [130], a human average yawn is 5 seconds and according to [131], the average adult yawns 20 times in a day, and it is assumed that as humans become more and more drowsy, their yawn duration becomes longer. Therefore, we estimate an average yawn of 20 times in 16 hours since we likely do not yawn when sleeping, since yawning, according to [132] is a way of cooling down the brain and not just a respiratory function as it was previously thought to be. We then estimate that an average human would yawn at most 2 times in an hour. This formulates our first condition:

   Condition 1: A person is deemed to exhibit signs of drowsiness if he yawns more than 2 times in a space of 5 minutes to 1 hour.

2. According to [133], the total number of eye blinks for a normal average person is 10 to 15 blinks per minute, and when a person is drowsy, the number of blinks becomes less, and the number becomes about 4 to 6 blinks per minute. This formulates our second condition:

   Condition 2: A person is deemed to exhibit signs of drowsiness if the number of blinks in a minute is 6 or less.

3. According to [134], eye blinks last just about a tenth of a second. If the blink lasts much longer than 200ms to 400ms [135], then, the person probably has his or her eyes closed, which is a strong indicator of drowsiness. This formulates our third condition:

   Condition 3: A person is deemed to exhibit signs of drowsiness if the eyes are closed for more than one second.

Based on these conditions highlighted above, we penalize the present CAV with p points, where:

1. $p = 2$, if according to condition 1, a person yawns more than 2 times in a space of 5 minutes to 1 hour. Thus, for each yawn more than 2 yawns between 5 minutes and hour or less, there is a penalty of 2 points.

2. $p = 5$, if according to condition 2, a person's eye blinks are only about 4 or 6 or less in a minute. Thus, for every minute, if the blink is not more than 6, there is a penalty of 5 points.

3. $p = 10$, if according to condition 3, a person's eyes are closed for more than one second. Thus, if a person's eyes are closed for more than a second, the present alertness value is penalized by 10 points.

Unlike eye blinks, yawning is a temporal activity, and as such, when the output from our CNNs returns a prediction as yawning, we do not just consider the decision as yawning, rather, we keep a track of the "yawning" predictions and if there are sequential yawning predictions for about 3 to 6 seconds, then, we deem the driver actually yawning. This is to minimize errors due to misclassification from our CNNs. It is worthy of note that the CAV and $\hat{CAV}$ are non-negative values, thus, when the CAV or $\hat{CAV}$ is zero, the penalties no longer apply.

5.9 Evaluation of Architecture

In section 5.2 to 5.6, we show the results of several convolutional neural network architectures which we used to train our dataset. To test the performance of our overall model, we use 5 videos [136], [137], [138], [139] and [140] which were gotten from YouTube. We manually cleaned some of these test data by removing frames where the driver is not showing. We also blurred out other faces in some videos where there are more than one person present. To create our ground truth, we manually go over the videos and took note of the facial information, with regards to the conditions set in Section 5.8. At the end of each video, we record our CAV, which is regarded as the ground truth. These same videos are then fed to the driver alertness framework using the different CNN architectures and compute the "Video Error". In this section, we discuss the performance of the driver alertness monitoring framework based on each CNN architecture in terms of "Video Error" and "Speed". The video error is modelled as the percentage error between the actual current alertness value (CAV) and the network's current alertness value $\hat{CAV}$, as described in Equation 4.7.

5.9.1 LeNet-5

The LeNet-5 was trained at different learning rates and with different pooling types. We also trained the network with batch normalization before and after activation and without batch normalization. Our findings showed that using LeNet-5 with batch normalization before activation, with average pooling as the pooling type, at a learning rate of 0.001 performed best, and we select this architecture as our final LeNet-5 architecture. With a face detected at about 0.26 seconds by the Dlib, the average time taken to process one frame up to the face-state prediction (normal, yawning or drowsy) is about 0.52 seconds. The LeNet-5 had the lowest performance based on each video error and average video error on the 5 test videos. The video errors (VE) on the 5 test videos were: $0.17, 0.2, 0.09, 0.15,$ and 0.13, with an average video error (AVE) of 0.148.

5.9.2 AlexNet

The driver alertness framework takes about 0.97 seconds to process an image when AlexNet is used for the face-state prediction. The video errors (VE) on the 5 test videos were: $0.15, 0.16, 0.06, 0.15,$ and 0.1 with an average video error (AVE) of 0.124.

5.9.3 VGG-16

The VGG-16 like AlexNet is fed with the same input and takes up to an average of 1.77 seconds to process a new frame up to the face-state prediction. The VGG-16 performed better than the LeNet-5 and AlexNet. The video errors (VE) on the 5 test videos were: $0.15, 0.15, 0.04, 0.13,$ and 0.11, with an average video error (AVE) of 0.116.

5.9.4 Inception-ResNet V2

The Inception-ResNet V2 seems to be the most performing model of all 5 in terms of the average video error (AVE). The Inception-ResNet V2 processes an image up to the face-state prediction for about 0.89 seconds, which means we process up to an image every second. The video errors (VE) on the 5 test videos were: $0.08, 0.06, 0.06, 0.06,$ and 0.05, with an average video error (AVE) of 0.062.

5.9.5 ResNext-50

The ResNext-50 is the second best model in terms of the average video error (AVE). The ResNext-50 falls short of the Inception-ResNet V2 by 0.006, however, the ResNext-50 model was able to process a frame up to the face-state prediction for about half the time it takes the Inception-ResNet to process a frame. The ResNext-

50 model which processes an image for about 0.56 seconds. The video errors (VE) on the 5 test videos were: $0.1, 0.08, 0.06, 0.05$, and 0.05, with an average video error (AVE) of 0.068.

5.10 Model Selection

In section 5.9, we evaluated the performance of the driver alertness monitoring models in terms of speed of processing and accuracy. The best model in terms of AVE is the Inception-ResNet V2, followed by the ResNext-50 model. However, in some videos, ResNext-50 competed favorably in terms of the VE. In addition, the ResNext-50 is much faster in processing new images than the Inception ResNet V2, as it processes an image about half the time it takes the Inception ResNet V2. After a careful comparison of the two best performing models, we selected the ResNext model as the overall best performing model with fast frames processing, competitive VEs with the Inception ResNet V2 and an AVE lesser by 0.006 than the Inception ResNet V2. In Table 5.9, we show the comparison of the results of the 5 models in terms of their accuracy on the 1880 test images, average image computation speed, VEs, and $AVEs$.

Architectures	Accuracy on Test Images (%)	Image Computation Speed (seconds)	Video Errors (VE)					Average Video Error (AVE)
			Vid 1	Vid 2	Vid 3	Vid 4	Vid 5	
LeNet-5	87.29	0.26	0.17	0.2	0.09	0.15	0.13	0.148
AlexNet	88.2	0.97	0.15	0.16	0.06	0.15	0.1	0.124
VGGNet	91.0	1.77	0.15	0.15	0.04	0.13	0.11	0.116
Inception ResNet V2	97	0.89	0.08	0.06	0.06	0.06	0.05	0.062
ResNext-50	96	0.56	0.1	0.08	0.06	0.05	0.05	0.068

Table 5.9: Comparing results of five trained CNN variants on 5 test videos.

5.11 Conclusion

In this Chapter, we presented the results obtained from our driver alertness framework using 5 convolutional neural networks variants which are: LeNet-5, AlexNet, VGGNet, Inception ResNet V2, and ResNext-50. We further discussed the results of these CNN variants under different sections. In Section 5.2, we discuss the performance of LeNet-5 when trained with batch normalization before and after activation, and also tuned other hyperparameters such as pooling type and learning rate and proved that hyperparameter tuning is a big deal and the right hyperparameters can improve a model's performance greatly. Section 5.3 discusses AlexNet's performance on the same dataset and we presented the performance plots, confusion matrix as well as a table showing the test accuracy. Section 5.4, 5.5 and 5.6 present the performances of VGGNet, Inception ResNet V2 and ResNext-50 respectively. We also showed their performance plots to see how the model trained over 100 epochs, and presented their performances on a test set of 1880 samples. We presented further findings in Section 5.7, where we discussed the effect of batch normalization and pooling type on LeNet-5. In Section 5.8, we present the drowsiness level evaluator which makes use of specific conditions to penalize the current alertness value of a driver. We also specified the thresholds for different levels of drowsiness in the same

section. Finally, Section 5.9 presents the comparison of the results of the driver alertness framework using the different CNN variants, and in Section 10, we present our best overall model which is the ResNext-50 and we provide a strong motivation for this, even though, it falls short by an average video error of 0.006. Our experiments and findings are not without challenges, and in Chapter 6, we discuss this in details.

Chapter 6

Discussion, Further Studies and Conclusion

In the previous Chapter, we present our results from the neural networks and selected our best performing model - ResNext-50, with a strong motivation of why we considered ResNext-50 the best model suited for this task. Our motivation majorly focuses on the faster image computation rate of ResNext-50 compared to the Inception ResNet V2 with best average video error. We considered this a good decision because the ResNext-50 only fell short by an average video error of 0.006, and competed favourably in terms of video errors in some scenarios.

In this chapter, we discuss our challenges while undergoing this research, and also suggest further studies and recommend what can be done to improve this research.

6.1 Challenges

6.1.1 Dataset

Although, we were able to obtain dataset from publicly available databases and also through Bing and Google APIs, most of our dataset are not robust enough to all scenarios - for example, yawning with the hand covering the mouth. Also, we did a lot of data augmentation because of our small dataset size. Although, we had videos from which we could generate thousands of frames, we only selected randomly from these samples as many of the frames were only repetitive of the other frames. Also, there are other scenarios of drowsy drivers with the faces tilted down, thus signifying that head pose might be a strong indicator of drowsiness. However, the data samples used to train our networks did not include this.

6.1.2 Speed of Processing

Although, we chose ResNext-50 as our best performing model, the time taken to process an image can be improved, and this would lead to better real time detection and predictions. There are quite a lot of features which are not unnecessary in our input images, as such, there are a lot of redundant weights in the final weight used for prediction. One way to combat this problem is through knowledge distillation of neural networks as proposed in [37] or by pruning the weights.

6.1.3 Time and Device Constraints

Due to time constraints, we were unable to generate more datasets to improve the performance of our architectures. We used Google and Bing APIs to scrap images relating to our study and manually filtered thousands of images and cleaned the needed ones, and we could have done included more data if we had more time. Also, we used K-1 fold cross validation while training our models, and this is also due to the time constraints we had, as some of the models trained for days on our computer. We were also constrained by device usage, as we only had access to computer systems with the specifications listed in Chapter 4. If we had access to systems with better compute power, we would have been able to better tune hyperparameters to improve our models' performance.

6.1.4 Drowsiness Conditions

In Chapter 5, we discussed 3 conditions used to track drowsiness levels in drivers, however, we believe there might be more conditions, and since we have limited access to sleep experts, we have only made use of the conditions we found in literature. However, while testing our framework with videos, we observed that head pose can be a strong indicator of drowsiness. We also found out from literature that humans tend to yawn for a longer duration when drowsy, and although, we found out from materials on the internet that an average human yawn is six seconds, we could not find a credible reference.

6.2 Further studies and recommendation

6.2.1 Sequence Prediction

Although, our framework keeps track of sequential information on input faces, and our model performs well, we however, think that using deep recurrent neural networks to solve the task of driver alertness monitoring might be more effective. This is because, there won't be a need to manually specify conditions or thresholds as the network will automatically learn drowsiness features by itself and probably much better. Convolutional neural networks are undoubtedly the best performing networks on images, and as such, it would be good to have CNNs help with feature extraction, thus we can combine CNNs with recurrent neural networks, for example, CNN-Long Term Short Term networks (CNN-LSTM) and we might obtain a better performance. However, to approach this problem with sequence modelling, a large database of drowsy drivers need to be made available. It would be best appropriate to gather dataset from drivers who drive long distances, thus, the giving a network the opportunity to learn how people transition from normal state to drowsy state.

We also recommend future researchers to look into using 3D Convolution for sequence modelling, as 3D convolution architectures are another great sequence modelling architectures, for example, [141] proposed using 3D convolution for human action recognition.

6.2.2 Time to Drowsiness

In this research, we only focused on monitoring driver alertness by predicting their current alertness value and level of drowsiness, however, driver alertness monitoring would be more efficient if we include a feedback as to how long it would take to get to a specific level of drowsiness, we call this Time to Drowsiness (TTD), this way, drivers are not only told what level of drowsiness they are currently on, but, rather make them know that they probably would be dangerously drowsy at a particular estimated time, this will in turn help smart vehicles to help know when to "forcefully" take over driving from a drowsy driver.

6.3 Packaged Model and Code

Our codes and results are available at our Bitbucket repository: jherrhy/wits-msc-research. You can send us an email at 1744903@students.wits.ac.za to have access to the repository.

6.4 Conclusion

In this research, we present a robust framework for a real time non-intrusive driver alertness monitoring system which was developed using deep convolutional neural networks to detect face-states through visual information, and using these face-states to evaluate drowsiness level. The motivation behind this research is that from our findings, we observed that a lot of lives have been lost to drowsiness related accidents. According to a report by [17], the amount of vehicular accidents would be reduced by 10% to 20% if drivers and driving behaviours can be monitored in real time and given warnings based on the analysis of driver behaviour. We thus, investigated several deep convolutional neural networks, and how hyperparameters can adjusted in order to obtain very high performance in terms of speed and accuracy. Since, this is a real time detection problem, there is a need to evaluate the models not just in terms of their accuracy, but also, speed of processing.

We developed a three-stage framework for the driver alertness monitoring system.

The first stage is face detection using the Dlib HoG based face detector which was cascaded with the Multi Task Cascaded Convolution Neural Network (MTCNN), and is only given an input image x_i, if only the Dlib HoG face detector is unable to find a face in the image x_i, which can be due to occlusion or high non-frontal position of the face. The face detector produces a bounding box of the face in the image which is then used to extract the face and serve it as input to the second stage.

The second stage of our frameworks is the face-state prediction, where we trained different convolutional neural network variants (such as LeNet-5, AlexNet, VGGNet, Inception ResNet V2 and ResNext-50) on a three classification output: Normal, Drowsy and Yawning. These face-states were extracted from Closed Eyes in the Wild (CEW) and YawDD databases. We also scraped images from the internet using Google and Bing APIs and manually filtered and cleaned them. We further did hyperparameter tuning to investigate the effect

of learning rate, batch normalization and pooling type on a network's performance using the LeNet-5, as it was easy and fast to train. After our networks have been successfully trained on the three face-states, we then feed a total of 1880 test image samples to evaluate their performances. Finally, we feed 5 videos through our framework (beginning from face detection) through to the face-state predictions and feed the output prediction to the third stage (which is the final stage of our framework).

The third stage is a decision layer which takes in to consideration decisions from the CNNs in the second stage and use a set of conditions to make decisions about a driver's current alertness level. The evaluator does this by penalizing the current alertness value (which at the beginning is always set to 100%) with points based on the conditions discussed in Chapter 5. We presented the performances of the framework with different CNNs used in the face-prediction state layer and selected the best performing model, which is the ResNext-50 and this is largely due to its faster computation speed compared to the Inception ResNet V2, which gave a better average video error and test accuracy on the 1880 samples. We selected the ResNext as the best overall model because, it competed favourably with the Inception-ResNet V2 in terms of video errors and only fell short of Inception-ResNet V2 architecture's average video error by 0.006. The ResNext-50 was able to process images at about half the time Inception-ResNet V2 does, and since our study focuses on real time monitoring, we found a model which has a good balance between speed and accuracy.

Finally, we consider this study a success as we are able to achieve all research objectives and have all our research questions answered. Our results show that monitoring drowsiness by keeping track of the predictions is a better and more effective way to monitor drowsiness than just making predictions with individual images.

Bibliography

[1] "World health organization road accident reports worldwide." https://www.who.int/gho/road_safety/mortality/en/. Accessed: 2019-02-10.

[2] "World health organization road accident reports worldwide 2020." https://www.asirt.org/wp-content/uploads/2020/03/road-traffic-injuries-the-facts-infographicen.pdf. Accessed: 2020-04-20.

[3] "World health organization road accident reports worldwide." https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries. Accessed: 2019-02-10.

[4] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[8] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[9] M. Ferguson, R. ak, Y.-T. Lee, and K. Law, "Automatic localization of casting defects with convolutional neural networks," pp. 1726–1735, 12 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[12] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[13] "World health organization road accident reports south africa." https://www.who.int/violence_injury_prevention/road_safety_status/2013/country_profiles/south_africa.pdf?ua=1. Accessed: 2019-02-10.

[14] "Drowsy driving nhtsa reports." https://www.nhtsa.gov/risky-driving/drowsy-driving. Accessed: 2019-02-10.

[15] "Distracted driving nhtsa reports." https://www.nhtsa.gov/risky-driving/distracted-driving. Accessed: 2019-02-10.

[16] A. S. Kulkarni and S. B. Shinde, "A review paper on monitoring driver distraction in real time using computer vision system," in *Electrical, Instrumentation and Communication Engineering (ICEICE), 2017 IEEE International Conference on*, pp. 1–4, IEEE, 2017.

[17] M. Bayly, B. Fildes, M. Regan, and K. Young, "Review of crash effectiveness of intelligent transportation systems," *Traffic Accident Causation in Europe (TRACE) deliverable D*, vol. 4, 2007.

[18] "Volvo driver assist control." https://www.volvocars.com/en-ca/support/cars?pc=y381&my=2015&sw=14w20&tab=ownersmanualonline&category=-om-en-ca-y381-2015-14w20&article=223155afaa5566f7c0a801e80181d286-om-en-ca-y381-2015-14w20. Accessed: 2019-02-12.

[19] "Mercedes attention assist." https://www.mercedes-benz.co.za/passengercars/mercedes-benz-cars/models/gle/gle-suv/explore/technologies-highlights/attention-assist.pi.html/mercedes-benz-cars/models/gle/gle-suv/explore/beosound-intro/beosound-detail. Accessed: 2019-02-12.

[20] "Bmw active driving assistant." https://www.press.bmwgroup.com/global/article/detail/T0141144EN/bmw-model-upgrade-measures-taking-effect-from-the-summer-of-2013. Accessed: 2019-07-24.

[21] "Audi rest recommendation system." https://www.audi-mediacenter.com/en. Accessed: 2019-07-24.

[22] "Ford driver alert." https://web.archive.org/web/20110513232258/http://media.ford.com/article_print.cfm?article_id=34562. Accessed: 2019-07-24.

[23] "Honda driver attention monitor." https://owners.honda.com/vehicles/information/2017/CR-V/features/Driver-Attention-Monitor. Accessed: 2019-07-24.

[24] "Volkswagen fatigue detection system." https://www.volkswagen.com.au/en.html. Accessed: 2019-07-24.

[25] "Bosch drowsiness detection system." https://www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/driver-assistance-systems/driver-drowsiness-detection/. Accessed: 2019-07-24.

[26] "Mazda driver attention alert." https://www.mazda.com/en/innovation/technology/safety/active_safety/daa/. Accessed: 2019-07-24.

[27] F. Rohit, V. Kulathumani, R. Kavi, I. Elwarfalli, V. Kecojevic, and A. Nimbarte, "Real-time drowsiness detection using wearable, lightweight brain sensing headbands," *IET Intelligent Transport Systems*, vol. 11, no. 5, pp. 255–263, 2017.

[28] V. Kartsch, S. Benatti, D. Rossi, and L. Benini, "A wearable eeg-based drowsiness detection system with blink duration and alpha waves analysis," in *Neural Engineering (NER), 2017 8th International IEEE/EMBS Conference on*, pp. 251–254, IEEE, 2017.

[29] C.-S. Wei, Y.-T. Wang, C.-T. Lin, and T.-P. Jung, "Toward drowsiness detection using non-hair-bearing eeg-based brain-computer interfaces," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 26, no. 2, pp. 400–406, 2018.

[30] A. Hashemi, V. Saba, and S. N. Resalat, "Real time driver's drowsiness detection by processing the eeg signals stimulated with external flickering light," *Basic and clinical neuroscience*, vol. 5, no. 1, p. 22, 2014.

[31] C.-Y. Wang, R. C.-H. Chang, C.-H. Lin, and S.-H. Su, "Fatigue detection system using enhanced so and chan method," in *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pp. 1–2, IEEE, 2018.

[32] C. J. de Naurois, C. Bourdin, A. Stratulat, E. Diaz, and J.-L. Vercher, "Detection and prediction of driver drowsiness using artificial neural network models," *Accident Analysis & Prevention*, 2017.

[33] H. Malik, F. Naeem, Z. Zuberi, and R. ul Haq, "Vision based driving simulation," in *Cyberworlds, 2004 International Conference on*, pp. 255–259, IEEE, 2004.

[34] J. Krajewski, D. Sommer, U. Trutschel, D. Edwards, and M. Golz, "Steering wheel behavior based estimation of fatigue," 2009.

[35] K. Mattsson, "In-vehicle prediction of truck driver sleepiness: lane position variables," 2007.

[36] P. Deram, "Vehicle based detection of inattentive driving for integration in an adaptive lane departure warning system," 2004.

[37] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, and J. Jang, "Real-time driver drowsiness detection for embedded system using model compression of deep neural networks," *Comput. Vis. Pattern Recognit. Work*, 2017.

[38] V. Saini and R. Saini, "Driver drowsiness detection system and techniques: a review," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 4245–4249, 2014.

[39] M. J. Flores, J. M. Armingol, and A. de la Escalera, "Real-time drowsiness detection system for an intelligent vehicle," in *Intelligent Vehicles Symposium, 2008 IEEE*, pp. 637–642, IEEE, 2008.

[40] S. Junaedi and H. Akbar, "Driver drowsiness detection based on face feature and perclos," in *Journal of Physics: Conference Series*, vol. 1090, p. 012037, IOP Publishing, 2018.

[41] R. Jabbar, K. Al-Khalifa, M. Kharbeche, W. Alhajyaseen, M. Jafari, and S. Jiang, "Real-time driver drowsiness detection for android application using deep neural networks techniques," *arXiv preprint arXiv:1811.01627*, 2018.

[42] R. Putta, G. N. Shinde, and P. Lohani, "Real time drowsiness detection system using viola jones algorithm," *International Journal of Computer Applications*, vol. 95, no. 8, 2014.

[43] A. Sayeed and S. A. Sadi, "Driver drowsiness detection using face monitoring and pressure measurement," *Journal of Embedded System & Applications*, vol. 5, no. 3, pp. 12–18, 2018.

[44] S. S. Bharambe and P. Mahajan, "Implementation of real time driver drowsiness detection system," *International Journal of Science and Research*, vol. 4, pp. 2202–2206, 2015.

[45] K. Kumari, "A real time driver drowsiness detection system,"

[46] A. Rahman, M. Sirshar, and A. Khan, "Real time drowsiness detection using eye blink monitoring," in *Software Engineering Conference (NSEC), 2015 National*, pp. 1–7, IEEE, 2015.

[47] C.-Y. Lin, P. Chang, A. Wang, and C.-P. Fan, "Machine learning and gradient statistics based real-time driver drowsiness detection," in *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pp. 1–2, IEEE, 2018.

[48] B. Manu, "Facial features monitoring for real time drowsiness detection," in *Innovations in Information Technology (IIT), 2016 12th International Conference on*, pp. 1–4, IEEE, 2016.

[49] T. Danisman, I. M. Bilasco, C. Djeraba, and N. Ihaddadene, "Drowsy driver detection system using eye blink patterns," in *Machine and Web Intelligence (ICMWI), 2010 International Conference on*, pp. 230–233, IEEE, 2010.

[50] L. Celona, L. Mammana, S. Bianco, and R. Schettini, "A multi-task cnn framework for driver face monitoring," in *2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pp. 1–4, IEEE, 2018.

[51] B. Hariri, S. Abtahi, S. Shirmohammadi, and L. Martel, "A yawning measurement method to detect driver drowsiness," *Technical Papers*, 2012.

[52] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association*, 2010.

[53] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[54] T. M. Mitchell *et al.*, "Machine learning. wcb."

[55] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education

Limited,, 2016.

[56] G. E. Hinton, T. J. Sejnowski, and T. A. Poggio, *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[57] X. J. Zhu, "Semi-supervised learning literature survey," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2005.

[58] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[59] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*, vol. 398. John Wiley & Sons, 2013.

[60] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, 2001.

[61] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling, "Feature selection using linear classifier weights: interaction with classification models," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 234–241, ACM, 2004.

[62] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE transactions on systems, man, and cybernetics*, no. 4, pp. 580–585, 1985.

[63] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[64] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[65] C.-L. Liu, "High accuracy handwritten chinese character recognition using quadratic classifiers with discriminative feature extraction," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2, pp. 942–945, IEEE, 2006.

[66] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for adaboost," *Machine learning*, vol. 42, no. 3, pp. 287–320, 2001.

[67] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[68] L. Breiman and J. H. Friedman, "Predicting multivariate responses in multiple linear regression," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 59, no. 1, pp. 3–54, 1997.

[69] J. L. Peixoto, "Hierarchical variable selection in polynomial regression models," *The American Statistician*, vol. 41, no. 4, pp. 311–313, 1987.

[70] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[71] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[72] Y. Sahin and E. Duman, "Detecting credit card fraud by ann and logistic regression," in *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pp. 315–319, IEEE, 2011.

[73] G. Guo, S. Z. Li, and K. Chan, "Face recognition by support vector machines," in *Proceedings fourth IEEE international conference on automatic face and gesture recognition (cat. no. PR00580)*, pp. 196–201, IEEE, 2000.

[74] M. Song and D. Civco, "Road extraction using svm and image segmentation," *Photogrammetric Engineering & Remote Sensing*, vol. 70, no. 12, pp. 1365–1371, 2004.

[75] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–I, IEEE, 2001.

[76] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in Deep Learning*. Springer, 2020.

[77] R. Dechter, *Learning while searching in constraint-satisfaction problems*. University of California,

Computer Science Department, Cognitive Systems ..., 1986.

[78] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[79] M. H. Hassoun, *Fundamentals of artificial neural networks*. MIT press, 1995.

[80] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[81] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[82] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[83] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted gaussian mixture models," *Digital signal processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

[84] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 129–136, 2011.

[85] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The" wake-sleep" algorithm for unsupervised neural networks," *Science*, vol. 268, no. 5214, pp. 1158–1161, 1995.

[86] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani, "Training generative neural networks via maximum mean discrepancy optimization," *arXiv preprint arXiv:1505.03906*, 2015.

[87] T. Virtanen, A. Mesaros, T. Heittola, M. D. Plumbley, P. Foster, E. Benetos, M. Lagrange, *et al.*, "Proceedings of the detection and classification of acoustic scenes and events 2016 workshop (dcase2016)," 2016.

[88] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[89] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pp. 643–650, ACM, 2015.

[90] C.-H. Weng, Y.-H. Lai, and S.-H. Lai, "Driver drowsiness detection via a hierarchical temporal deep belief network," in *Asian Conference on Computer Vision*, pp. 117–133, Springer, 2016.

[91] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[92] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[93] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[94] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[95] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *arXiv preprint arXiv:1604.03168*, 2016.

[96] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of the twenty-first international conference on Machine learning*, p. 18, ACM, 2004.

[97] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[98] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in neural information pro-*

cessing systems, pp. 2654–2662, 2014.

[99] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2014.

[100] B. B. Sau and V. N. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," *arXiv preprint arXiv:1610.09650*, 2016.

[101] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.

[102] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*, pp. 319–345, Springer, 1999.

[103] "Cs231n.github.io. cs231n convolutional neural networks for visual recognition.." http://cs231n.github.io/convolutional-networks/. Accessed: 2019-02-12.

[104] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[105] Q. Liang, J. Mu, M. Jia, W. Wang, X. Feng, and B. Zhang, "Communications, signal processing, and systems," in *Conference proceedings CSPS*, p. 3, 2017.

[106] S. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing*, vol. 378, pp. 112–119, 2020.

[107] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[108] F. Song, X. Tan, X. Liu, and S. Chen, "Eyes closeness detection from still images with multi-scale histograms of principal oriented gradients," *Pattern Recognition*, vol. 47, no. 9, pp. 2825–2838, 2014.

[109] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," in *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[110] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[111] X. Tan, F. Song, Z.-H. Zhou, and S. Chen, "Enhanced pictorial structures for precise eye localization under incontrolled conditions," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1621–1628, IEEE, 2009.

[112] "Closed eyes in the wild." http://parnec.nuaa.edu.cn/xtan/data/ClosedEyeDatabases.html. Accessed: 2019-02-10.

[113] R. Fusek, "Pupil localization using geodesic distance," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11241 LNCS, pp. 433–444, 2018.

[114] "Yawning detection dataset." http://www.site.uottawa.ca/~shervin/yawning/. Accessed: 2019-02-10.

[115] S. Abtahi, M. Omidyeganeh, S. Shirmohammadi, and B. Hariri, "Yawdd: A yawning detection dataset," in *Proceedings of the 5th ACM Multimedia Systems Conference*, pp. 24–28, ACM, 2014.

[116] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[117] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[118] S. Yang, P. Luo, C.-C. Loy, and X. Tang, "Wider face: A face detection benchmark," in *Proceedings of*

the IEEE conference on computer vision and pattern recognition, pp. 5525–5533, 2016.

[119] M. Everingham, A. Zisserman, C. K. Williams, L. Van Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó, *et al.*, "The 2005 pascal visual object classes challenge," in *Machine Learning Challenges Workshop*, pp. 117–176, Springer, 2005.

[120] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," 2015.

[121] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *2014 IEEE international conference on image processing (ICIP)*, pp. 343–347, IEEE, 2014.

[122] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*, pp. 437–478, Springer, 2012.

[123] Y. Bengio, A. Courville, and I. J. Goodfellow, "Deep learning: adaptive computation and machine learning," *Bengio. A. Courville*, 2016.

[124] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[125] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[126] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[127] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[128] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[129] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[130] J. Askenasy, "Is yawning an arousal defense reflex?," *The Journal of psychology*, vol. 123, no. 6, pp. 609–621, 1989.

[131] "Why are yawns contagious." https://www.pbs.org/newshour/science/why-are-yawns-contagious-we-asked-a-scientist. Accessed: 2020-02-23.

[132] "Why do we yawn." https://www.independent.co.uk/news/science/why-do-we-yawn-nothing-to-do-with-boredom-were-just-cooling-our-brains-say-scientists-9 html. Accessed: 2020-02-23.

[133] M. Raha and A. Roy, "Fatigue estimation through face monitoring and eye blinking," in *International conference on mechanical, industrial and energy engineering*, 2014.

[134] "How fast is a blink of an eye." https://sciencing.com/fast-blink-eye-5199669.html. Accessed: 2020-04-19.

[135] U. Svensson, *Blink behaviour based drowsiness detection: method development and validation*. Statens väg-och transportforskningsinstitut, 2004.

[136] "Test video." https://www.youtube.com/watch?v=KwR_rwzNJ38. Accessed: 2020-04-19.

[137] "Test video." https://www.youtube.com/watch?v=jpT1jlqODwg. Accessed: 2020-04-19.

[138] "Test video." https://www.youtube.com/watch?v=TrfIN-GZr3g. Accessed: 2020-04-19.

[139] "Test video." https://www.youtube.com/watch?v=ccJ7aZrQ3SA. Accessed: 2020-04-19.

[140] "Test video." https://www.youtube.com/watch?v=jpT1jlqODwg. Accessed: 2020-04-19.

[141] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.