MASTER OF TECHNOLOGY

PATTERN RECOGNITION SYSTEMS (PRS)

Driver Alertness Detection System

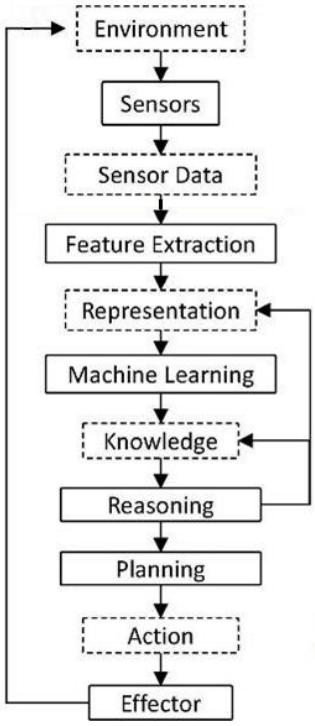
Group 7

Name	Student ID
Hwang Sion	A0249263Y
Prerak Agarwal	A0116711R
Santi	A0249294R
Zhang Junfeng	A0249266U

Table of Contents

EXECUTIVE SUMMARY

This project intends to build a complete working prototype for a driver alertness detection system with the aim of improving the safety of drivers on the road. As covered in the ISSM module, this is an end-to-end Intelligent Pattern Recognition and Sense Making System implementation. Below is a pipeline of system implementation, with explanations on how our project components fit into each bucket.



Stage	Elements in Project
Environment	Inside a car being driven by a driver.
Sensors	Front video camera on driver's mobile device placed in front of the driver (for instance, while being used for navigation). In context of this project – camera attached to a Raspberry Pi.
Sensor Data	Live stream of video frames (images) captured through the camera.
Feature Extraction	Individual frames processed through convolutional layers.
Representation	The outputs from the convolutional layers.
Machine Learning	All the different models we built to recognise the alertness state of the driver (CNN, CNN-RNN, etc.).
Knowledge	After applying the models on the input data, the system would know whether the driver is being alert while driving or not.
Reasoning	An added layer of logic by humans to specify that drowsiness and/or distraction is not good while driving.
Planning	Preparing to issue an immediate alert for the driver if drowsiness or distraction is detected.
Action	Issuing an alert to the driver (could be in the form of a LED light or a sound).
Effector	The LED light being lit up and/or the sound being played.

As a group of four, we explored a variety of approaches to build our system effectively. Throughout this journey, we explored and built 14 machine learning models, and gained invaluable knowledge and insights on how machine learning techniques can be implemented into working systems. Following are some of the main areas we spent a significant amount of effort and gained experience in through practical system implementation:

- **Data sourcing and preparation** for training and testing.
- **Building different types of models** to solve for the same goal.
- **Model tuning**: Exploring ways to increase accuracy and reduce overfit.
- **Transfer learning using pre-trained models**: Utilizing existing pre-trained models properly and building on top of them for more effective training.
- **Deployment on edge devices**: Implementing models that are trained on premium GPUs (for instance, ones available in Google Colab Pro+) into edge devices (like the Raspberry Pi).

BUSINESS PROBLEM BACKGROUND

Year on year, there has been a notable increase in the number of car accidents, especially in Singapore. According to the annual statistics released by the Traffic Police of Singapore, the number of fatal accidents had risen by 25% from 2020 to 2021 (Shiying, 2022).

Year	Total Motor Vehicle Population	Accidents resulting in Casualties	Fatal Accidents	Fatality Rate per 100,000 persons
2010	945,829	8,808	188	3.80
2011	956,704	8,597	192	3.76
2012	969,910	8,184	162	3.16
2013	974,170	7,748	150	2.96
2014	972,037	7,959	150	2.83
2015	957,246	8,206	148	2.75
2016	956,430	8,417	140	2.51
2017	961,842	7,843	117	2.16
2018	957,006	7,810	120	2.20
2019	973,101	7,822	117	2.07

Figure 1. Year by Year motor vehicle casualties/fatal accident rate (EvlanovaAnastassia, 2021)

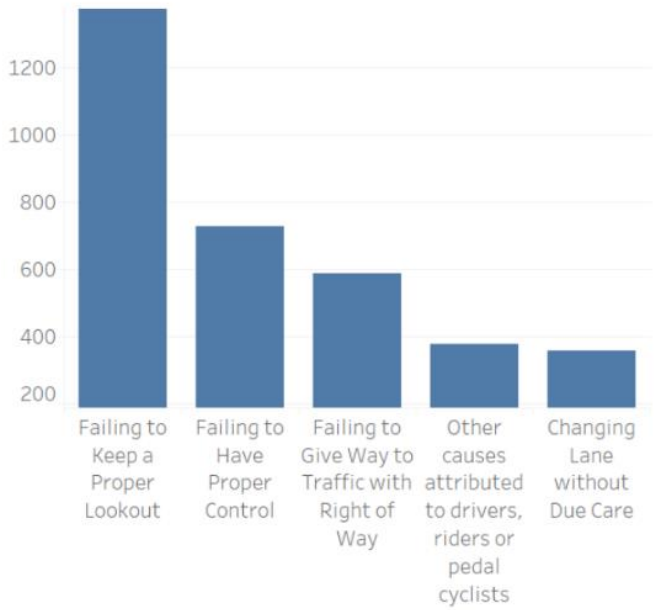


Figure 2. Top 5 Reasons for Car Accidents in Singapore (ShayedAmeer, 2020)

Among the top causes for this increase in the number of accidents on Singapore roads were 'failure to keep a proper lookout' and 'failure to have proper control of the vehicle' (Singapore Police Force, 2020). Some causes that may have led to such accidents could include external causes of distraction such as operating the phone while driving, talking to passengers, failure to lookout for slippery road conditions, etc., or internal causes such as fatigue due to prolonged driving.

Various solutions have been proposed to address the issue with the alertness of drivers while driving. A popular solution, known as the Driver Monitoring System (DMS), is fast becoming common and default in most new cars, with projections that it will become a standard safety feature as soon as 2023. (Lenne, 2021)

PROJECT OBJECTIVES & SUCCESS MEASUREMENTS

PROJECT OBJECTIVES

The objective of this project is to develop a system that helps to improve the safety of drivers on the road, especially private hire car drivers who drive for long hours of the day at a stretch. The system, relying on computer vision capabilities and built using deep learning techniques, should be able to detect signs of drowsiness and/or distraction in drivers in real-time and alert them automatically in such cases. The idea behind developing such a system is to improve the safety of drivers who spend a major part of their day driving, with the intention being that such a system could eventually be incorporated as a feature in the drivers' apps of ride-hailing services such as Grab, Gojek, etc., among others.

Most drivers of ride-hailing services already position their smartphones in front of them while driving and use them for navigation. Keeping this in mind, the idea for our safety system is to continuously monitor the video live-feed from the front camera of the driver's smartphone for signs of drowsiness and/or distraction. If the computer vision model detects such signs, an alert would be issued to the driver immediately. The goal for this project is to prove its feasibility as a potential safety feature to be incorporated in the drivers' apps of ride-hailing services (such as Grab, Gojek, etc.).

SUCCESS MEASUREMENTS

With the above objectives outlined, we can define a few design considerations below that may be used as a measure of success for our project.

1. **Should issue real-time alerts:** The safety system should monitor the driver's video live-feed and issue alerts in real-time.

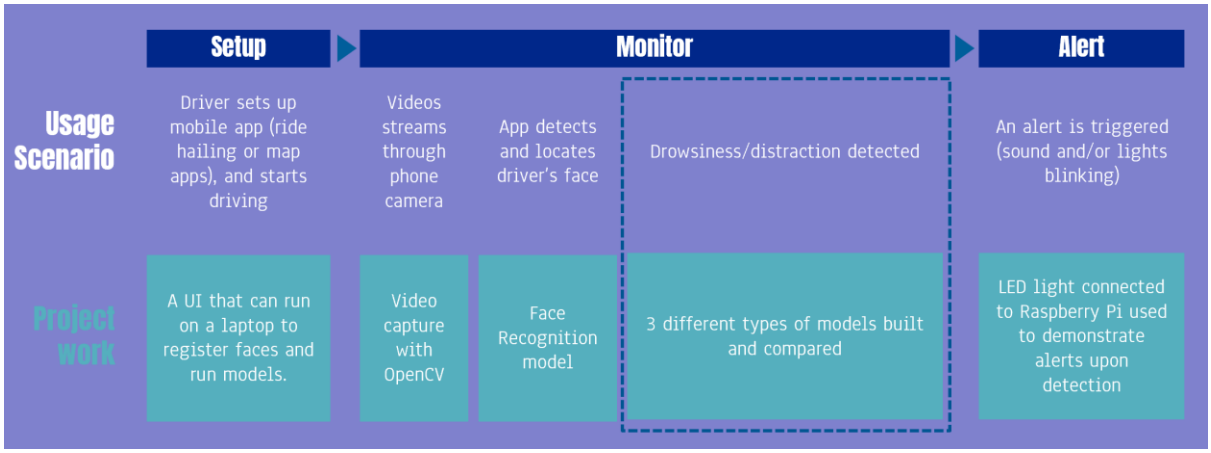
2. **Deployable on edge devices:** The safety system, once implemented, should be able to run entirely locally on the driver's device without the need for any of the video data to be transmitted to any other device/server over the internet. This will help ensure that any privacy concerns arising from the continuous monitoring of the driver's video live-feed are not violated.

3. **Should only monitor the driver:** The live-feed video frame from the front camera of the driver's smartphone may include people other than the driver (such as the passengers). In such a scenario, only the driver's face should be considered by the computer vision model to detect signs of drowsiness and/or distraction.

PROJECT SOLUTION DESIGN AND TECHNIQUES

SYSTEM ARCHITECTURE

The system is design to fulfil the usage scenario which spans across setup, monitor and alert stages.



User Setup

It starts from the driver start the mobile app and start driving. The assumption here is that the user (the driver) has registered their profile photos with the app so that the face_recognition function can detect the driver's face. The ideal final implementation of the model is to be part of a mobile app with navigation function. It can be taxi hailing app such as Grab, Uber or Gojek. It can also be map apps like Google map. In these apps, especially the hailing apps, the app registers the user's role as the driver, and takes photo of the driver. All we need to do is to add this alertness detection as a function that turns on when the driver is on the road.

In our project, we don't do the integration with any of the mobile apps mentioned above due to limitation in time and resources. However, we developed a user interface to perform the setup and use the function. This demonstrates that the solution can be implemented as a complete working system.

Monitor

While the driver is being monitored all the time. If it is detected that the driver is drowsy or not focused on driving, the system sends an alert. In our implementation, we focused on build the models. We tried different approaches in building models to monitor the driver's face and actions. Some models focus on monitoring facial landmarks to determine if the driver is yawning or becoming sleepy; some other models are trained to detect the driver's actions directly frame by frame, understanding if the driver is driving normally or distracted or drowsy; apart from frame-by-frame detection models, we also built models that takes in a sequence of frames to perform prediction.

Alert

The purpose of this solution is to provide timely alert when needed so that the driver focuses back to driving. Alerting is also part of the functions we developed. In our implementation, we use an LED light attached to Raspberry Pi to demonstrate alerting. This can be easily changed to playing a sound for example. It is all about triggering a signal.

TECHNOLOGY STACK

Technology stack differentiated into 2 categories, stack that being used for model training and stack that being used for deployment

	Training	Deployment
Software	**Python** • Tensorflow • Keras • Numpy • Sklearn • Matplotlib • Pandas • Seaborn • PathLib • pygame • Face_Recognition • imutils • OpenCV • Tensorflow-metal to support GPU in macOS	**Python** • Tflite-runtime • Numpy • Facial-recognition • OpenCV • Tkinter • dlib
Hardware	**Google Colab Pro+ Subscription** • **RAM**: 90 GB • **GPU**: NVIDIA-SMI 460.32.03; Driver Version: 460.32.03; CUDA Version: 11.2; GPU Memory size: 40 GB **Apple Mac Studio- M1 Max** • **RAM**: 64 GB • **GPU**: 32 cores GPU with Metal GPUFamily Apple 7 **Google Colab Standard Subscription** • **RAM**: 13.6 GB • **GPU**: NVIDIA-SMI 460.32.03; Driver Version: 460.32.03; CUDA Version: 11.2; GPU Memory size: 14.75 GB **MSI Laptop** • **RAM**: 32 GB • **GPU**: NVIDIA GeForce RTX 3070 • **Processor**: 11th Gen Intel(R) Core(TM) i9-11900H @ 2.50GHz	**Raspberry Pi 4 Model B** • **Storage** (SD card): 32 GB • **RAM**: 8 GB **Extension to Raspberry Pi:** • GPIO Cable • Breadboard • Breadboard Adopter • LED Light with Resistor • USB Camera

MACHINE LEARNING MODELS

There are different ideas to achieve the objective of the project. The core of the system is the machine learning models that processes the captured video content streamed live through a camera and infer if the driver in the car is alert or not. We thought of the following approaches, and we have implemented each of them in this project.

S/N	Model Architecture	No of frames per inference	Classification Classes	Objective	Variations of Models Built in the Project
1	CNN	Per frame	1. Eyes open or closed 2. Mouth open ratio	To check if the driver is drowsy or not.	Different ways to create the model that we have tried 1. CNN Model trained from scratch and optimisation 2. Transfer learning with Xception Model 3. Transfer learning with InceptionV3 4. Transfer learning with ResNet50 5. Transfer learning with MobileNet
2	CNN	Per frame	1. Normal Driving 2. Texting with right hand 3. Talking to phone with right hand 4. Texting with left hand 5. Talking to phone with left hand 6. Turn on / actively working on radio 7. Drinking 8. Reaching Behind 9. Talking to passenger 10. Playing with hair / make up 11. Being drowsy	To check if the driver is not focusing on driving.	Different ways to create the model that we have tried 1. CNN Model trained from scratch and optimisation 2. Transfer learning: MobileNetV2 with 10 classes 3. Transfer learning: MobileNetV2 with 9 classes 4. Transfer learning: MobileNetV2 with 11 classes 5. Transfer learning: VGG16 with 9 classes
3	CNN + RNN	Sequence of Frames	1. Alert 2. Drowsy	To check if the driver is drowsy or not.	Different ways to create the model that we have tried 1. A ConvLSTM model trained from scratch (no transfer learning). 2. An LRCN (Long-term Recurrent Convolutional Network) model trained from scratch (no transfer learning).
4	CNN	Per frame	1. Not the pre-set user 2. The pre-set user	To check if the driver (pre-set up) is in the captured frame.	Two CNN models based on different libraries: 1. deepface 2. face_recognition

RASPBERRY PI DEPLOYMENT DESIGN

To demonstrate edge device deployment, we chose Raspberry Pi for the following reasons:

1. It is small enough to represent an edge device. This requires us to make the models small enough to be deployed on such devices.
2. It is a full-fledged computer with a Linux OS and a desktop UI, USB ports and even HDMI ports. This way, we can easily attach a camera through the USB port. HDMI port also comes in handy during the setup stage.
3. It has an GPIO panel that can extend the integration with other devices or peripherals. This opens many possibilities for the demo. In our implementation, we used an LED light to show the signal when drowsiness is detected.

SYSTEM DEVELOPMENT, PERFORMANCE VALIDATION AND DISCUSSIONS

In this driver alertness detection system, there are 2 main characteristics the team focused into:

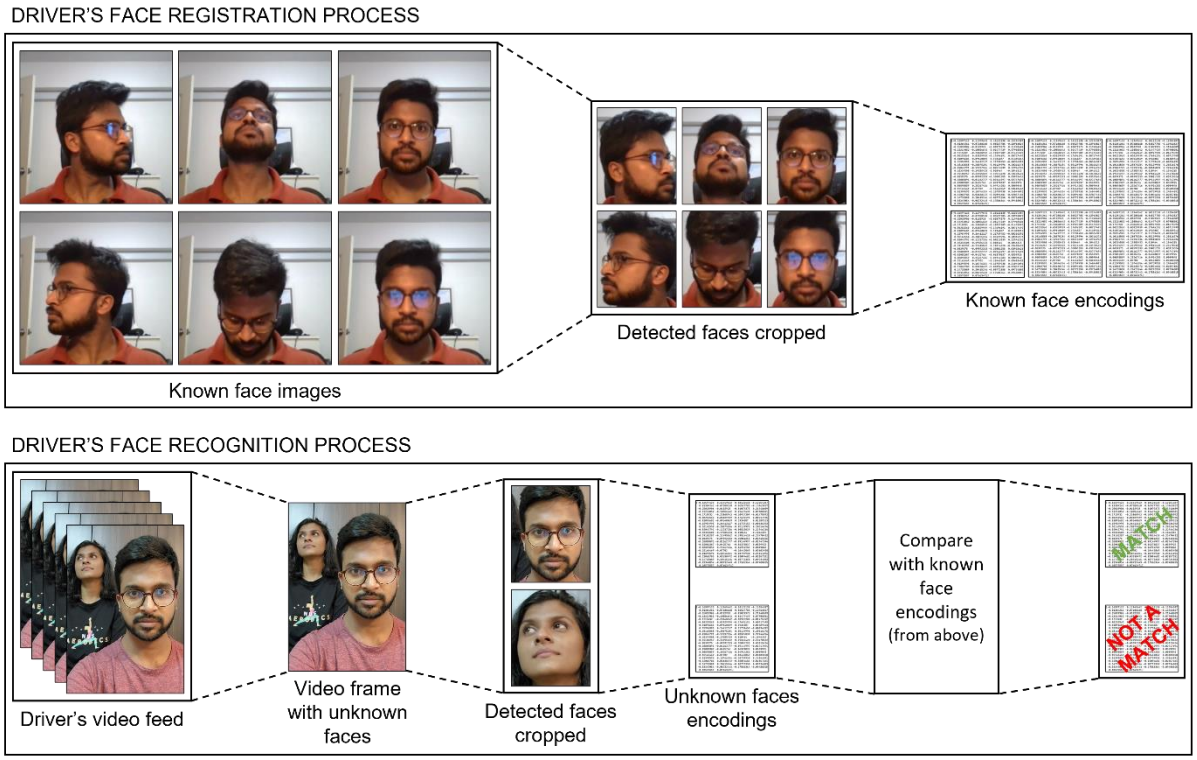
1. Driver is not alert due to drowsiness
2. Driver is not alert due to performing other activities

FACIAL RECOGNITION FUNCTION (FOR DRIVER IDENTIFICATION)

As mentioned in the Success Measurements section above, one of the design considerations for our project was to ensure that our system only monitors the drivers face for signs of distraction and drowsiness. In the real world, there may be scenarios where the live-feed video frame from the front camera of the driver's smartphone includes faces of people other than the driver, such as the passengers. In such scenarios, it would not be logical to detect signs of drowsiness and distraction in those other faces and issue an alert (for instance, it would not be logical to issue a drowsiness alert if the passenger is sleeping while the driver is awake). It is important for the computer vision model to only consider the driver's face to detect signs of drowsiness and distraction.

To meet this requirement, we explored a few different options to recognise the driver's face. The idea here is that the system would perform facial recognition on the captured video frames, and only if the driver's face is detected in the video frames would these frames be passed into the alertness detection models.

To perform the facial recognition task, we explored two different Python libraries namely face_recognition (Geitgey, 2022) and deepface (Serengil, 2022). The approaches for the facial recognition task with both the libraries were similar and are explained below in greater detail.



In both cases, the first step in the process is for the user (the driver) to register themselves by providing a few pictures of their face (referred to as the 'known face'). The model would detect faces in all the supplied images and crop the images accordingly. These cropped pictures would then be used to extract features from the known face and create a set of known 'facial encodings' (to be used later in the process). The next step would then be to supply the model with a new image that contains a face (referred to as the 'unknown face'). This image would be the video frame from the driver's front-camera feed. Similar to the known face images above, the model would detect faces in the supplied video frame and crop the frame accordingly. It would then extract similar features from the unknown faces and compare them against the set of know facial encodings extracted above. If the two sets of encodings satisfy the match criteria (higher than a certain threshold), the model would return a True for match, else it would return False.

After executing this task with both the libraries, it was noticed that the face_recognition library generally utilised lesser computational power and performed quicker, albeit with slightly lesser accuracy. Since one of the key success measurements for this project is to make this safety system deployable on edge devices, face_recognition library was chosen for the final implementation.

APPROACH 1 – CNN MODEL BASED ON FACIAL LANDMARKS

CNN model was created by using eye face data from the UMass Amherst. Based on live data from the camera using OpenCV frames captured, this model identifies if user's eye is closed or open. If user's eye is closed for more than specified threshold, it will throw alert message with sound using pygame API to user to alert user. Blinking of eyes will not throw any alerts as system only sends out alerts when eye is closed consecutively for specified time.

Also, this model captures position of the mouth and if user's mouth is open for longer than specified threshold, it will make sound alert to user.

Using face landmark, this model identifies mouth position and alerts user. Lip shape between 48~60 points below is captured as well using shape predictor.

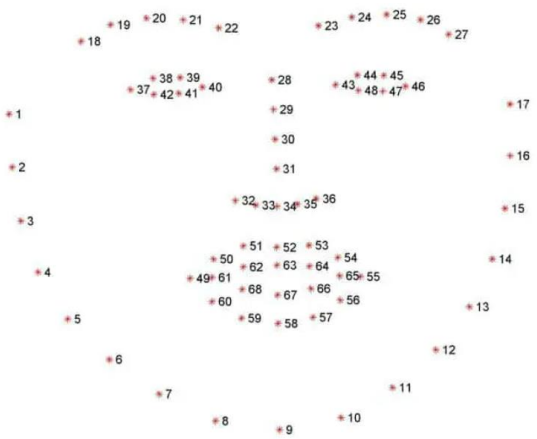
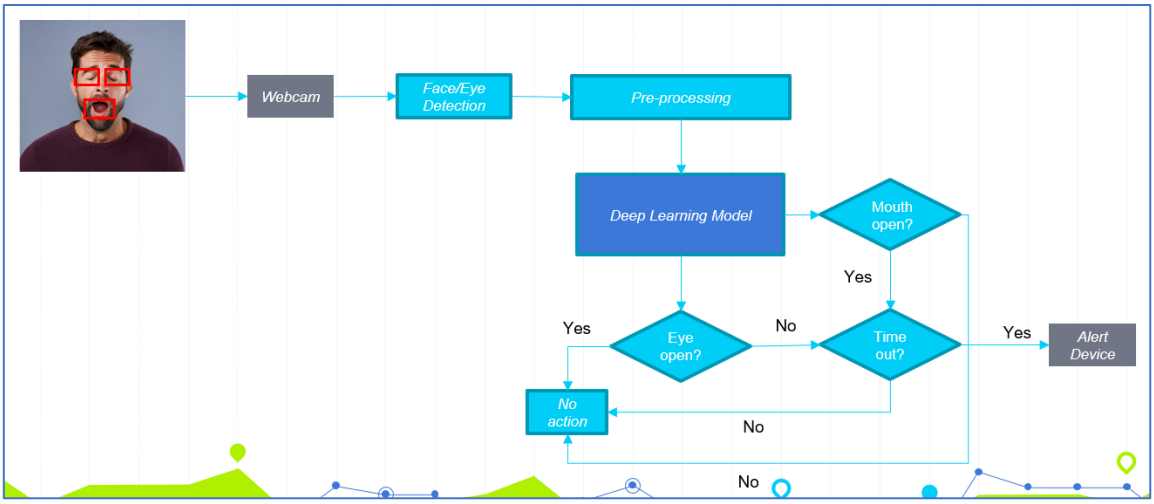


Figure 1: Visualization of the 68 facial landmark coordinates (RosebrockAdrian, 2017)

Our team has experimented by creating our own model first and tried to optimize model's accuracy as much as possible by optimizing various settings such as dropout rate, number of layers, batch normalization etc.



Also, to compare the performance with the best model that is built manually, we utilized existing deep learning Keras models created in TensorFlow and applied transfer learning. Keras application provide deep learning model which is readily available with pre-trained weights which can be used for prediction, extraction of features, and fine tuning. Weights are instantiated automatically when model is downloaded.

Our team has decided to adopt and compare our own model with transfer learning models which allows to retain the final layer of an existing model, decreasing training time as well as size of dataset required for training.

Below is the list of available deep learning models available in TensorFlow Keras:

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6

Table 1. List of Keras Applications (Keras, 2022)

Out of many deep learning models available, a few models are selected to identify best model for our use case. Consideration is given in choosing transfer learning models available such as top-1 and top-5 accuracy, number of parameters, size of model, depth of model, and time (ms) per inference step (GPU).

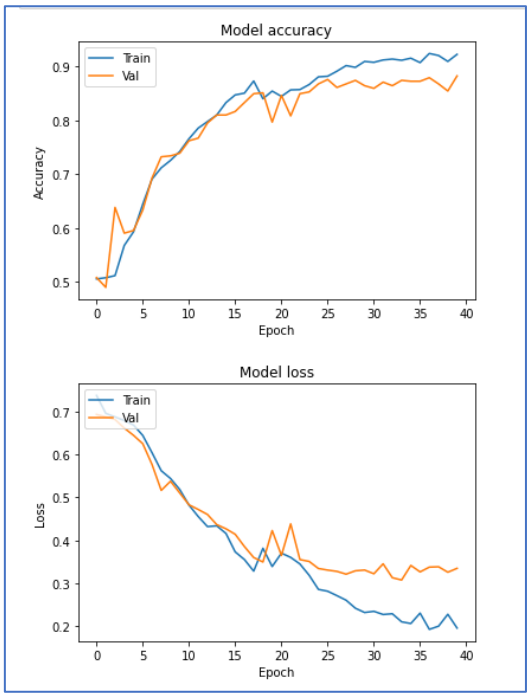
Below dataset is used as common dataset for all training models.

Data Source	Description	Type	Data Size (No. of Samples)	Data Size
UMass Amherst dataset	Open/Closed eye images	Images	1231 open eye Images and 1192 closed eye images with 24x24 pixels.	1MB

Team has selected following models for comparison:

1. **CNN Model created and trained from scratch by team**
   This model has 5 convolutional layers with 'ReLU' activation function for feature extraction and max pooling is applied. Sigmoid activation function is used in last layer as eye open status is either 'Open' = 1 or 'Closed' = 0.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 224, 224, 3)]     0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 224, 224, 32)      896
_________________________________________________________________
activation_5 (Activation)    (None, 224, 224, 32)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
activation_6 (Activation)    (None, 224, 224, 32)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
activation_7 (Activation)    (None, 224, 224, 32)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 112, 112, 32)      9248
_________________________________________________________________
activation_8 (Activation)    (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 112, 112, 32)      9248
_________________________________________________________________
activation_9 (Activation)    (None, 112, 112, 32)      0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 100352)            0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               25690368
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 65
=================================================================
Total params: 25,769,473
Trainable params: 25,769,473
Non-trainable params: 0
_________________________________________________________________
None
```



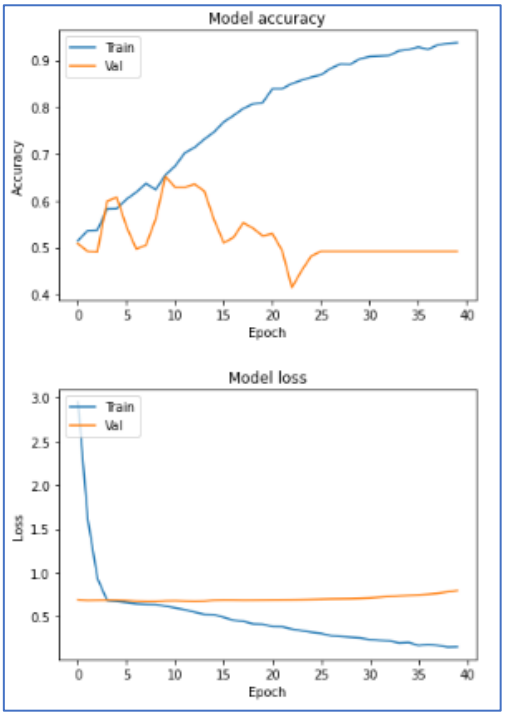
This model has proved to have pretty good validation accuracy of 88% which has proven to be sufficient for the actual model to run in Raspberry pi. Also, model didn't have any anomaly or spikes in terms of valuation or loss showing consistent increment of model accuracy.

```
Accuracy 0.8828382838283828
[[277  31]
 [ 40 258]]
              precision    recall  f1-score   support

           0       0.87      0.90      0.89       308
           1       0.89      0.87      0.88       298

    accuracy                           0.88       606
   macro avg       0.88      0.88      0.88       606
weighted avg       0.88      0.88      0.88       606
```

2. **CNN model with Batch Normalization**

From the 1st model, batch normalization and dropout are applied to further optimize the model. Batch normalization is applied to standardize the layer inputs for every mini-batch. We expected this will stabilize the learning process and reduce the number of epochs required to train deep network.

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 224, 224, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 224, 224, 32)      128
_________________________________________________________________
activation (Activation)      (None, 224, 224, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 224, 224, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
activation_1 (Activation)    (None, 224, 224, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 112, 112, 32)      128
_________________________________________________________________
activation_2 (Activation)    (None, 112, 112, 32)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 56, 56, 32)        128
_________________________________________________________________
activation_3 (Activation)    (None, 56, 56, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 32)        9248
_________________________________________________________________
activation_4 (Activation)    (None, 56, 56, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 6,502,273
Trainable params: 6,502,081
Non-trainable params: 192
```



Adding batch normalization reduced validation accuracy by 49% and validation accuracy didn't improve even with 40 epoch of training. So we have concluded that it is really case by case to use batch normalization to improve existing models.

```
Accuracy 0.49174917491749176
[[  0 308]
 [  0 298]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       308
           1       0.49      1.00      0.66       298

    accuracy                           0.49       606
   macro avg       0.25      0.50      0.33       606
weighted avg       0.24      0.49      0.32       606
```

3. Xception Model

Xception model which is created by google is abbreviation of 'Extreme version of Inception'. With a modified depth-wise separable convolution, it outperforms Inception-v3 which is also created by Google, which was 1st runner up of Large-Scale Visual Recognition Challenge (ILSVRC) in 2015 for both ImageNet ILSVRC and JFT datasets.

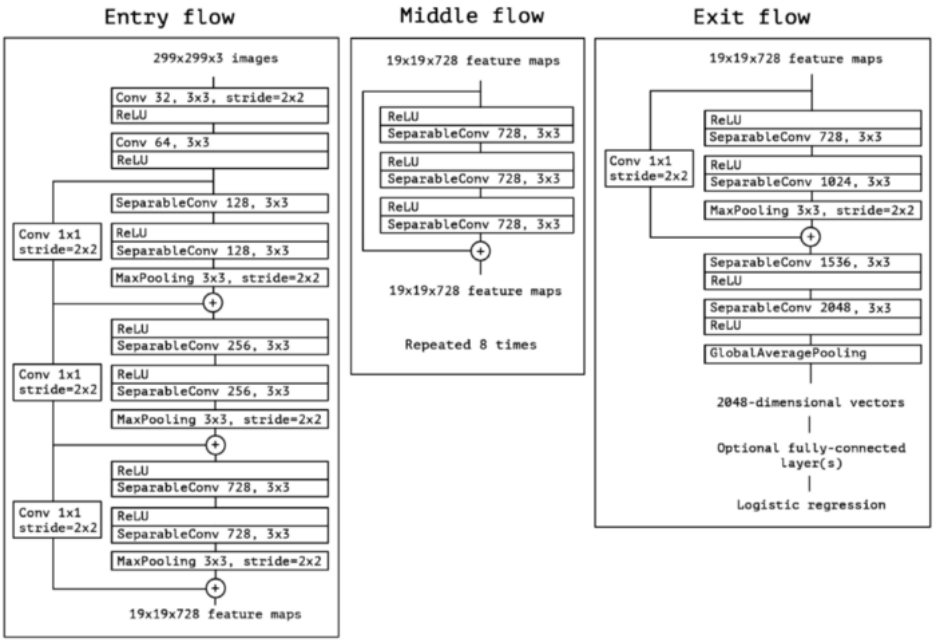


Figure 3. Overall Architecture of Xception

(TsangSik-Ho, Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification), 2018)

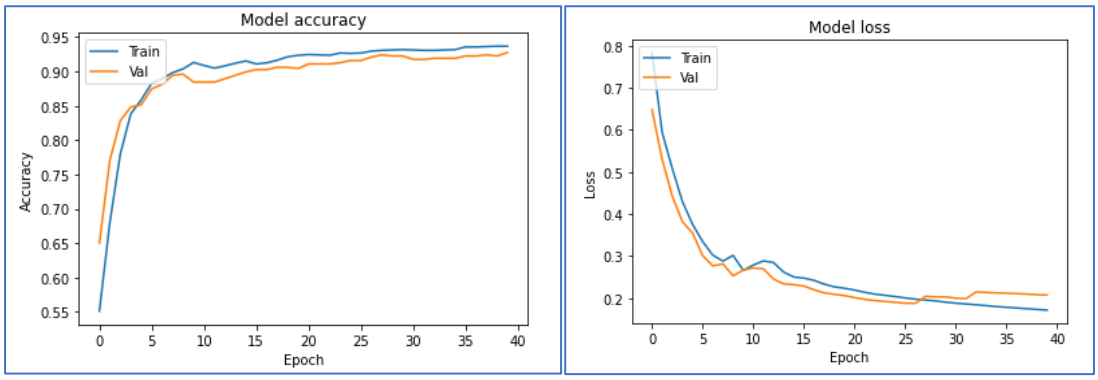
From Figure above for Xception model, SeparableConv is the modified depth-wise separable convolution which performs pointwise convolution first before doing a depth-wise convolution. These SeparableConv are used as inception modules throughout the deep learning architecture (TsangSik-Ho, Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification), 2018).

```
=================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________
```

Total number of parameters was above 20 million and only last few layers added was used as trainable parameters to utilize weights already learned in this model.

```
Accuracy 0.9273927392739274
[[285  23]
 [ 21 277]]
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       308
           1       0.92      0.93      0.93       298

    accuracy                           0.93       606
   macro avg       0.93      0.93      0.93       606
weighted avg       0.93      0.93      0.93       606
```

The model has managed to achieve very high validation accuracy of 92% which was highest amongst all the transfer learning models tried by the team.



Steady increment of the model accuracy and steady decrease in model loss was observed throughout the model training and validation as each epoch passes.

With the moderate level of depth (81 layers) and parameters (22.9M), this model has managed to achieve best result throughout. This is thanks to efficient architecture which is built with 2 main concepts: 1) Depth-wise Separable Convolution 2) Shortcuts link between Convolution blocks similar to ResNet implemenation.

4. Inception-v3

Inception-v3 is created by Google, which was 1st runner up of Large-Scale Visual Recognition Challenge (ILSVRC) in 2015. This model was initially trained with over a million images from 1,000 classes with very powerful machines.

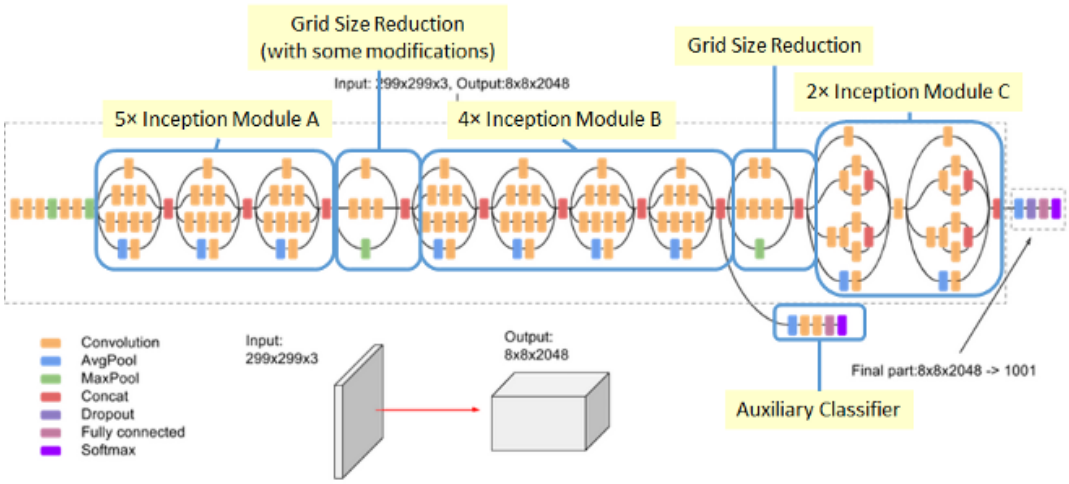


Figure 4. Inception-v3 Architecture

(TsangSik-Ho, Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015, 2018)

Inception-v3 is created by considering below main concepts (TsangSik-Ho, Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015, 2018):

1. Factorized Convolutions
   - Objective is to decrease number of connections/parameters without compromising on efficiency of the network.
2. Smaller Convolutions
   - Two 3×3 convolutions substitute one 5×5 convolution:
     - For 1 layer of 5×5 filter, number of parameters = 5 × 5 = 25
     - For 2 layers of 3×3 filters, number of parameters = 3 × 3 + 3 × 3 = 18
   - There is decrease in number of parameters by 28%.
3. Asymmetric convolutions

- A 3 × 3 convolution is replaced with 1 × 3 convolution and 3 × 1 convolution subsequently.

4. Auxiliary classifier
   - Small CNN is added between layers throughout training and loss created will be added to main network loss.
5. Grid size reduction
   - This is done mainly by pooling operations. Efficient network with less expensive cost is created with grid size reduction.

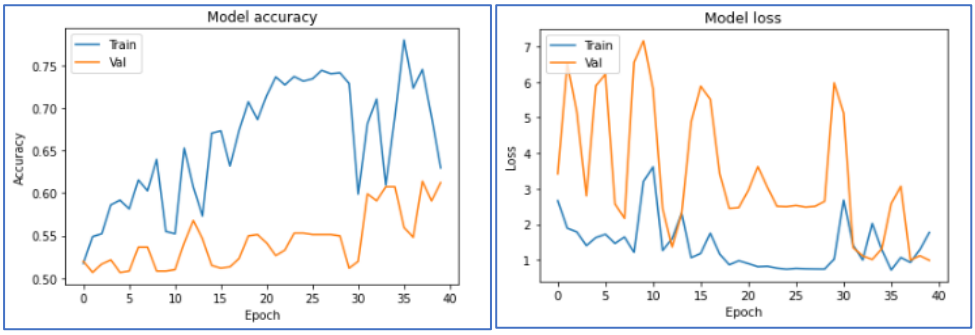
```
=================================
Total params: 21,804,833
Trainable params: 2,049
Non-trainable params: 21,802,784
_________________________________
```

Total number of parameters was above 21 million and only last few layers added was used as trainable parameters to utilize weights already learned in this model.

Batch Normalization and ReLU are used after Convolutional layers.

```
Accuracy 0.6122112211221122
[[189 119]
 [116 182]]
              precision    recall  f1-score   support

           0       0.62      0.61      0.62       308
           1       0.60      0.61      0.61       298

    accuracy                           0.61       606
   macro avg       0.61      0.61      0.61       606
weighted avg       0.61      0.61      0.61       606
```

The model has managed to achieve low validation accuracy of 61% which was significantly lesser than CNN model created by the team (88% validation accuracy) or Xception model (92% validation accuracy).



InceptionV3 model validation accuracy gradually increased but not so much improvement was obtained in terms of minimizing validation loss as epoch goes on and there was huge spike of loss in between the epochs showing that this model is not so suitable for our use case.

5. ResNet50

Residual Networks (ResNet) was created by Microsoft Research in 2015. Deep Neural Network trains with back propagation process with gradient descent and find the weights which minimize loss function. When there are too many layers, repeated multiplication will either reduce gradient until it disappears, causing performance to drop drastically (MukherjeeSuvaditya, 2022).

ResNet solves “Vanishing Gradient” problem with “Skip Connections”. ResNet piles up multiple convolution layers, and skips those layers, and use the activations of the previous

layer again. Skip connections increase initial training speed by compressing deep neural network into few layers (MukherjeeSuvaditya, 2022).

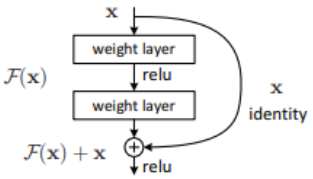


Figure 5. Residual learning building block
(MukherjeeSuvaditya, 2022)

Below is ResNet-50 Architecture diagram.

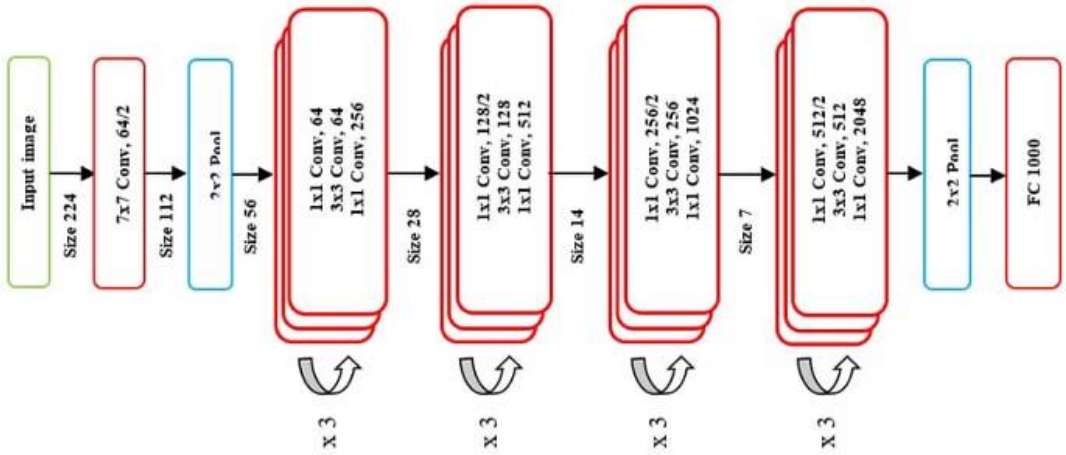


Figure 6. ResNet-50 Architecture
(MukherjeeSuvaditya, 2022)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				

Figure 7. ResNet-50 Structure
(MukherjeeSuvaditya, 2022)

The ResNet-50 architecture can be summarized into 6 parts (MukherjeeSuvaditya, 2022):

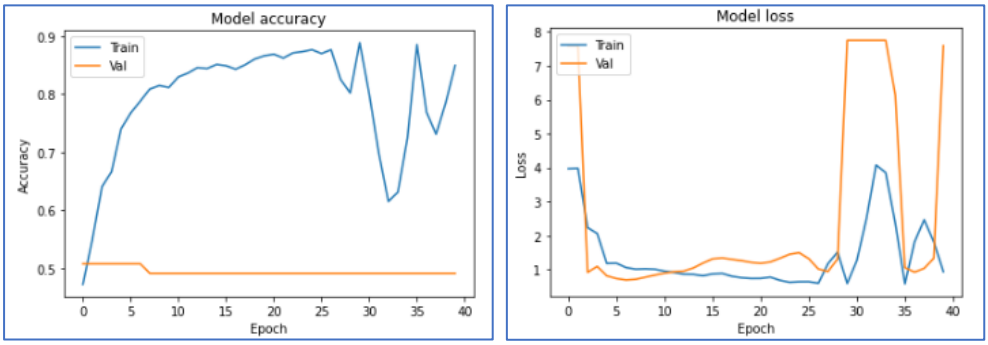
1. Input Pre-processing
2. Cfg[0] blocks
3. Cfg[1] blocks
4. Cfg[2] blocks
5. Cfg[3] blocks
6. Fully connected layer

```
=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
_________________________________________________________________
```

Total number of parameters was above 23 million and only last few layers added was used as trainable parameters to utilize weights already learned in this model.

```
Accuracy 0.49174917491749176
[[  0 308]
 [  0 298]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       308
           1       0.49      1.00      0.66       298

    accuracy                           0.49       606
   macro avg       0.25      0.50      0.33       606
weighted avg       0.24      0.49      0.32       606
```

ResNet-50 achieved very low validation accuracy of 49% which was worst of all transfer learning models tried.



Although training accuracy has improved over time to achieve up to 88%, validation accuracy never improved and remained around 49% which is clear indicator that this model was not suitable for our use case. Also, model loss has increased drastically between 31st to 35th epoch which makes the model not so reliable for usage.

6. **MobileNet**

MobileNet is created to be deployed into mobile applications. MobileNet uses depth-wise segregation of convolutions. This model drastically decreased number of parameters to train, resulting in lightweight deep neural network (PujaraAbhijeet, 2020).

A depth-wise separable convolution is created with two operations (PujaraAbhijeet, 2020).

- Depth-wise convolution
  - Depth-wise convolution is the channel-wise DK×DK spatial convolution. Suppose in the figure below, and we have three channels; then, we will have 3 Dk×Dk spatial convolutions.
- Point-wise convolution.
  - Point-wise convolution is the 1×1 convolution to change the dimension.

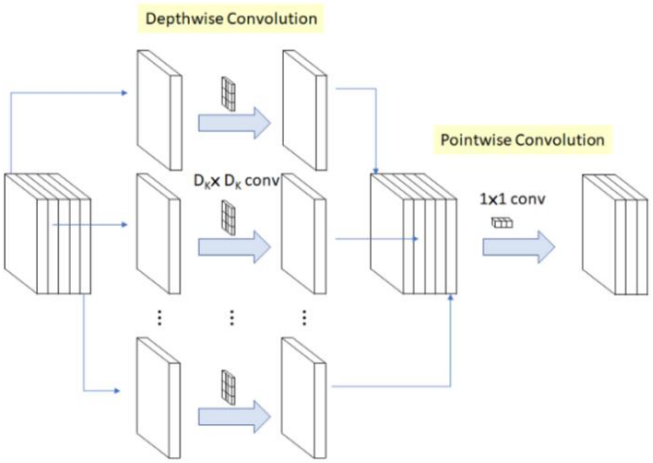


Figure 8. Depth-wise Separable Convolution
(PujaraAbhijeet, 2020)

Below is MobileNet Architecture diagram.

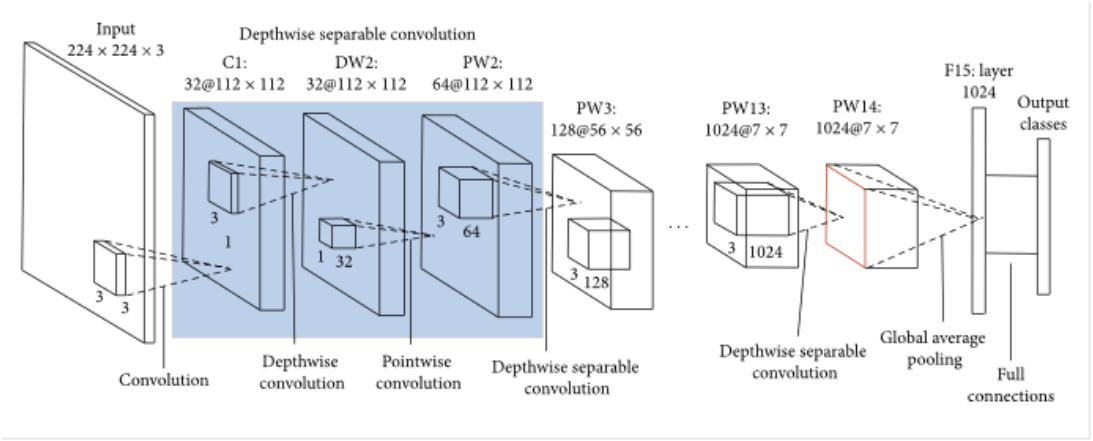


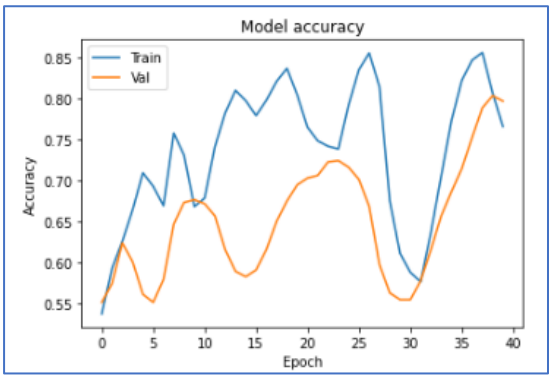
Figure 9. MobileNet Architecture
(PujaraAbhijeet, 2020)

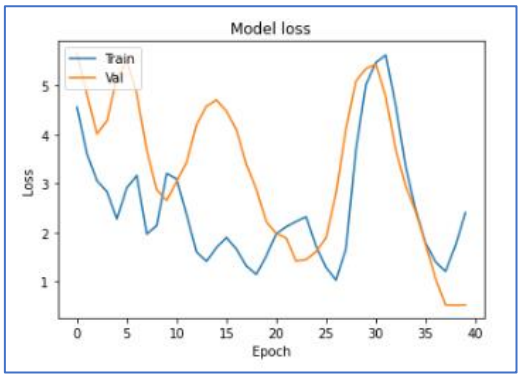
```
=================================
Total params: 3,229,889
Trainable params: 1,025
Non-trainable params: 3,228,864
_________________________________
```

Total number of parameters was above 3 million and only last few layers added was used as trainable parameters to utilize weights already learned in this model.

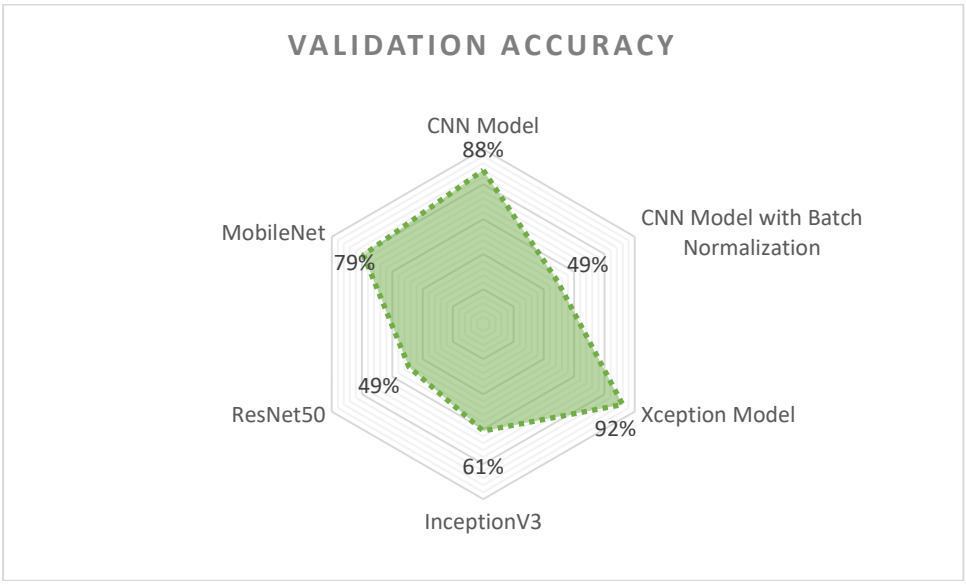
```
Accuracy 0.7970297029702971
[[265  43]
 [ 80 218]]
              precision    recall  f1-score   support

           0       0.77      0.86      0.81       308
           1       0.84      0.73      0.78       298

    accuracy                           0.80       606
   macro avg       0.80      0.80      0.80       606
weighted avg       0.80      0.80      0.80       606
```

MobileNet achieved moderate validation accuracy of around 80% which was slightly lower than CNN model created by the team (88% validation accuracy).





One of the observations from the model training is that training and validation accuracy goes through lots of ups and downs throughout the epoch. Eventually model validation accuracy has improved to reach around 80% but model seemed little inconsistent to be utilized on our use case.



All in all, Xception Model (validation accuracy of 92 %) and CNN model created by our team (validation accuracy of 88%) was best model for our use case in terms of model accuracy as well as stability of the model, thus these were chosen for our choice for deployment model.

APPROACH 2 – CNN FULL FACE & BODY POSTURE DETECTION MODEL

There are 2 datasets to build CNN model:

1. Dataset based on full face to indicate drowsy or alert
2. Dataset based on driver focus towards driving and the outside environment. There are many instances driver is losing focus due to some activities.

CNN Model based on full face feature

The dataset utilised for this model was obtained from National Tsing Hua University and is known as the Driver Drowsiness Detection Dataset (Computer Vision Lab, National Tsing Hua University, 2016). The dataset contains simulated driving videos of 36 people in a variety of scenarios, such as normal driving, yawning, slow blink rate, falling asleep, laughing, etc., with the total video time of the entire dataset being close to 9.5 hours. The labels for each video in this dataset are provided per video frame level and divided into two classes – 0 for ‘Stillness’ and 1 for ‘Drowsy’.

The dataset utilised for this model was obtained from National Tsing Hua University and is known as the Driver Drowsiness Detection Dataset (Computer Vision Lab, National Tsing Hua University, 2016). The dataset contains simulated driving videos of 36 people in a variety of scenarios, such as normal driving, yawning, slow blink rate, falling asleep, laughing, etc., with the total video time of the entire dataset being close to 9.5 hours. The labels for each video in this dataset are provided per video frame level and divided into two classes – 0 for 'Stillness' and 1 for 'Drowsy'.

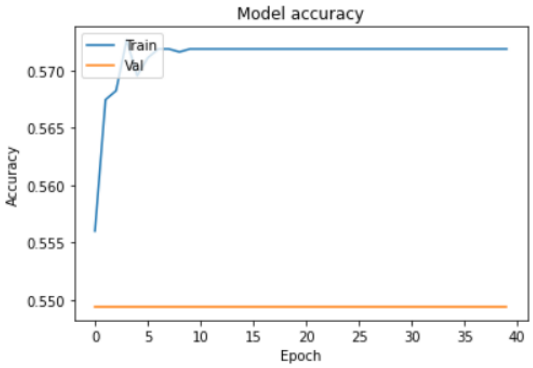
Data Source	Description	Type	Data Size (No. of Samples)	Data Size
National Tsing Hua University – Driver Drowsiness Detection Dataset	Dataset with 36 individuals in various simulated driving scenarios, including *normal driving, yawning, slow blink rate, falling asleep, etc.*, under day and night illumination conditions.	Videos	Training + Validation – 90 videos (1-1.5mins each, total 9.5hrs) Testing – 90 videos	± 7.45GB

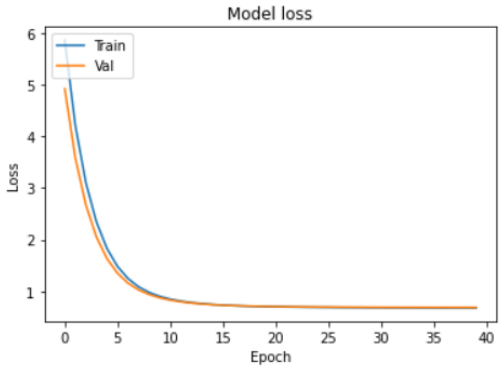
Data Preparation: To prepare this dataset for a CNN model and convert the video data to images, video frames were extracted randomly from the videos along with their corresponding labels. The image below shows a batch of extracted video frames along with their labels prior to model training.

Model building: We started the model building process with a simple model comprising of a few pairs of Conv2D and MaxPooling layers, followed by a few Dense layers. However, that simple model architecture resulted in the model not training at all. So, to make the model deeper, more layers were added iteratively. Different combinations of Dropout layers were also added iteratively to enable the model to train effectively and make it generalizable. Different layers for random data augmentation were also added after the Input layer to further help with model training. However,

despite all these efforts, the model did not train effectively, as seen in the accuracy and loss graphs below. It was concluded at this stage that the data across the two classes is very similar in nature, which makes it incredibly difficult for the model to train. To train better models, we then decided to try other techniques, such as merging our dataset with another dataset and transfer learning, as explained in the following sections.

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 240, 240, 1)]     0

random_flip_5 (RandomFlip)   (None, 240, 240, 1)       0

random_brightness_5 (Random  (None, 240, 240, 1)       0
Brightness)

random_rotation_5 (RandomRo  (None, 240, 240, 1)       0
tation)

conv2d_25 (Conv2D)           (None, 240, 240, 8)       40

max_pooling2d_25 (MaxPoolin  (None, 120, 120, 8)       0
g2D)

conv2d_26 (Conv2D)           (None, 120, 120, 32)      1056

max_pooling2d_26 (MaxPoolin  (None, 60, 60, 32)        0
g2D)

conv2d_27 (Conv2D)           (None, 60, 60, 128)       16512

max_pooling2d_27 (MaxPoolin  (None, 30, 30, 128)       0
g2D)

dropout_15 (Dropout)         (None, 30, 30, 128)       0

conv2d_28 (Conv2D)           (None, 30, 30, 32)        16416

max_pooling2d_28 (MaxPoolin  (None, 15, 15, 32)        0
g2D)

conv2d_29 (Conv2D)           (None, 15, 15, 16)        2064

max_pooling2d_29 (MaxPoolin  (None, 7, 7, 16)          0
g2D)

flatten_5 (Flatten)          (None, 784)               0

dense_25 (Dense)             (None, 256)               200960

dense_26 (Dense)             (None, 128)               32896

dense_27 (Dense)             (None, 32)                4128

dropout_16 (Dropout)         (None, 32)                0

dense_28 (Dense)             (None, 8)                 264

dropout_17 (Dropout)         (None, 8)                 0

dense_29 (Dense)             (None, 2)                 18

=================================================================
Total params: 274,354
Trainable params: 274,354
Non-trainable params: 0
_________________________________________________________________
```





CNN Model based on body posture:
Data is obtained from Kaggle (https://www.kaggle.com/c/state-farm-distracted-driver-detection). Data consist of 22424 (training and validation) and 79727 unlabelled test data. Data distribution is relatively balance for all class around 2200 – 2500. 5% of data is taken for testing (as testing data is not labelled). Subsequently during training 20% of the data for validation.

Data Source	Description	Type	Data Size (No. of Samples)	Data Size
Kaggle.com - State Farm Distracted Driver Detection Dataset	State-farm distraction dataset. The data consist of 10 classes such as normal driving, talking on the phone, texting, makeup, drinking and others.	Images	22,424 (training + validation) 79,727 (testing – unlabelled)	± 4.52 GB

There are 10 different classes in dataset

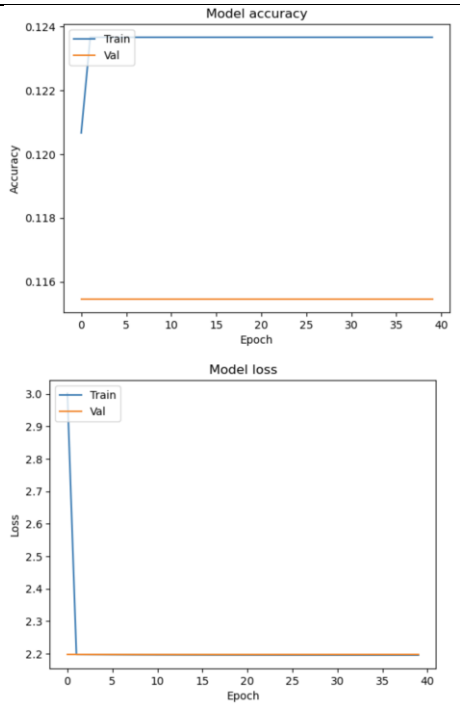
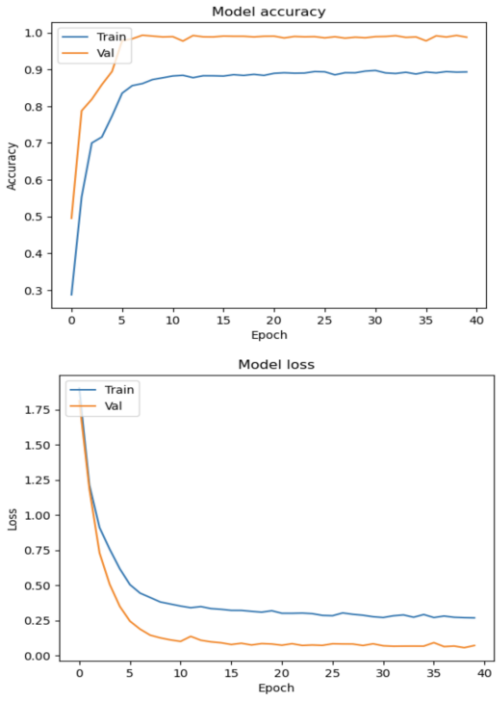
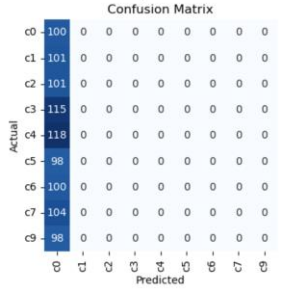
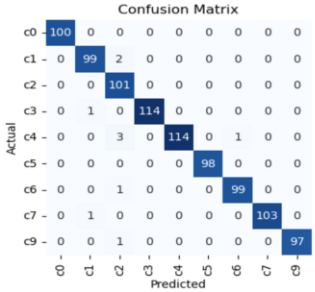
1. Normal Driving
2. Texting with right hand
3. Talking to phone with right hand
4. Texting with left hand
5. Talking to phone with left hand
6. Turn on / actively working on radio
7. Drinking
8. Reaching Behind
9. Playing with hair / make up
10. Talking to passenger

Data Preparation:
The training dataset had been labelled. The images are in RGB with dimension 640 * 480. In order to reduce the need of higher compute power and speed up training, we reduce the dimension into 224 * 224

Model building:
We started with simple model creation as below, with 2 layers, batch normalization, activation, max pooling. At the start we exclude touching with hair and makeup to avoid gender bias model.

The result are unusable / untrainable model which might due to large image and the layers created not enough to extract features, so there are various attempt being done to improve accuracy. Due to limited compute power, we are reading references on how to improve models without creating too huge models.
Addition of layers made the model total parameter more, so as we come across MobileNet we come across architecture where separable convolutional concept is used. There are two main types of separable convolutions: spatial separable convolutions, and depth-wise separable convolution.

The main difference between Depth-wise and normal convolution is: in the normal convolution, we are transforming the image x times of kernels. And every transformation uses up multiplications. In the separable convolution, we only really transform the image once (Wang, 2018) — in the depth-wise convolution. Then, we take the transformed image and simply elongate it to X channels. Without having to transform the image over and over again, we can save up on computational power.

Comparison	Initial model	After tuning
Architecture	Metal device set to: Apple M1 Max Model: "model" Layer (type) — Output Shape — Param # input_1 (InputLayer) — [(None, 224, 224, 3)] — 0 Input_conv (Conv2D) — (None, 222, 222, 32) — 896 layer1_bn (BatchNormalization) — (None, 222, 222, 32) — 128 Layer1_conv (Conv2D) — (None, 222, 222, 64) — 18496 layer2_bn (BatchNormalization) — (None, 222, 222, 64) — 256 Layer1_Blk_relu (Activation) — (None, 222, 222, 64) — 0 max_pooling2d (MaxPooling2D) — (None, 111, 111, 64) — 0 flatten (Flatten) — (None, 788544) — 0 dense (Dense) — (None, 32) — 25233440 dropout (Dropout) — (None, 32) — 0 dense_1 (Dense) — (None, 9) — 297 Total params: 25,253,513 Trainable params: 25,253,321 Non-trainable params: 192	Metal device set to: Apple M1 Max Model: "model" Layer (type) — Output Shape — Param # input_1 (InputLayer) — [(None, 224, 224, 3)] — 0 Input_conv (Conv2D) — (None, 222, 222, 32) — 896 layer1_bn (BatchNormalization) — (None, 222, 222, 32) — 128 depthwise_conv2d (DepthwiseConv2D) — (None, 222, 222, 32) — 64 Layer1_conv (Conv2D) — (None, 222, 222, 64) — 18496 layer2_bn (BatchNormalization) — (None, 222, 222, 64) — 256 depthwise_conv2d_1 (DepthwiseConv2D) — (None, 222, 222, 64) — 128 Layer1_Blk_relu (Activation) — (None, 222, 222, 64) — 0 max_pooling2d (MaxPooling2D) — (None, 111, 111, 64) — 0 flatten (Flatten) — (None, 788544) — 0 dense (Dense) — (None, 64) — 50466880 dropout (Dropout) — (None, 64) — 0 dense_1 (Dense) — (None, 9) — 585 Total params: 50,487,433 Trainable params: 50,487,241 Non-trainable params: 192
Accuracy and Loss		
Confusion Matrix	935/935 [==========] - 11s 12ms/step there were 100 correct predictions in 935 tests for an accuracy of 10.70 % 	935/935 [==========] - 11s 12ms/step there were 925 correct predictions in 935 tests for an accuracy of 98.93 % 

Classification Report	Best accuracy (on testing dataset): 10.70% precision recall f1-score support c0 0.1070 1.0000 0.1932 100 c1 0.0000 0.0000 0.0000 101 c2 0.0000 0.0000 0.0000 101 c3 0.0000 0.0000 0.0000 115 c4 0.0000 0.0000 0.0000 118 c5 0.0000 0.0000 0.0000 98 c6 0.0000 0.0000 0.0000 100 c7 0.0000 0.0000 0.0000 104 c9 0.0000 0.0000 0.0000 98 accuracy 0.1070 935 macro avg 0.0119 0.1111 0.0215 935 weighted avg 0.0114 0.1070 0.0207 935	Best accuracy (on testing dataset): 98.93% precision recall f1-score support c0 1.0000 1.0000 1.0000 100 c1 0.9802 0.9802 0.9802 101 c2 0.9352 1.0000 0.9665 101 c3 1.0000 0.9913 0.9956 115 c4 1.0000 0.9661 0.9828 118 c5 1.0000 1.0000 1.0000 98 c6 0.9900 0.9900 0.9900 100 c7 1.0000 0.9904 0.9952 104 c9 1.0000 0.9898 0.9949 98 accuracy 0.9893 935 macro avg 0.9895 0.9898 0.9895 935 weighted avg 0.9898 0.9893 0.9894 935
Remark		Added 2 depth-wise convolutional and increase dense layer resulted in a better model result.

Since the accuracy of model that we had tried not reach a satisfactory result, especially due to huge model size. We expand our experiment with transfer learning. There are 2 models we tried for the same datasets

1. MobilNetV2

As the implementation of this model is at mobile phone or other resource constraint device such as Raspberry Pi, one of the architectures being tested is MobilNetV2. The basic building block is bottleneck depth-separable convolutional with residuals. The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers. ReLU6 is used as the non-linearity because of its robustness when used with low-precision computation. Kernel size 3 × 3 as is standard for modern networks and utilize dropout and batch normalization during training. [MobileNetV2: Inverted Residuals and Linear Bottlenecks (https://arxiv.org/abs/1801.04381)]

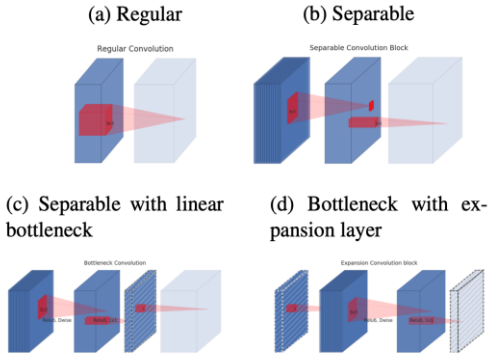


Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

[(Mark Sandler, 2019)MobileNetV2: Inverted Residuals and Linear Bottlenecks (https://arxiv.org/abs/1801.04381)]

2. VGG16

As the dataset obtained is quite large in dimension, we also experiment with VGG16 as it meant for large scale images. Following is some architectural description of VGG16. The input to VGG16 ConvNets is a fixed-size 224 × 224 RGB image. The only pre-processing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where it uses filters with a very small receptive field: 3 × 3

(which is the smallest size to capture the notion of left/right, up/down, centre) [(Karen Simonyan, 2015)]

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

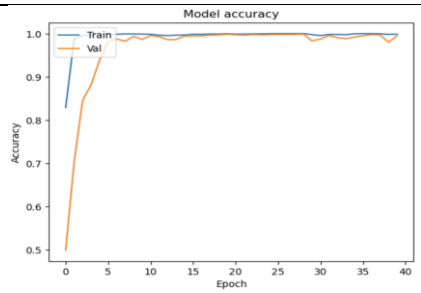
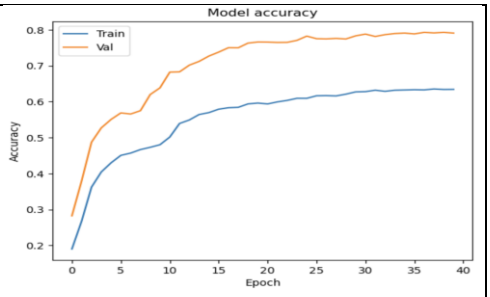
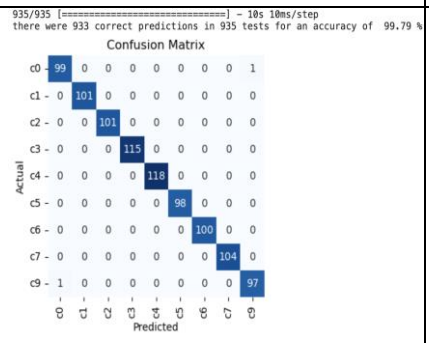
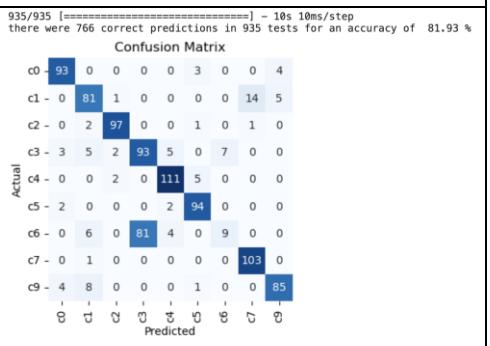
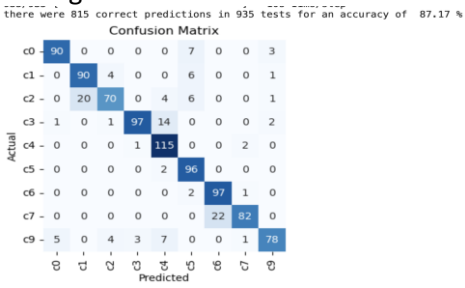
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

[VGG16: Very Deep Convolutional Networks for Large-Scale Image Recognition (Karen Simonyan, 2015)]

Based on training & test, the transfer learning result that we observed:

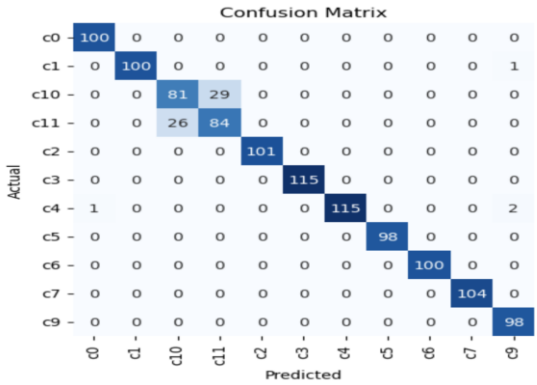
1. VGG16 slightly has higher accuracy but the model size is roughly 4x bigger than model from MobilNetV2
2. Setting some parameter as Non trainable made the model has reached lower accuracy at the same epoch, but it's speed up the training time. Below is the comparison and observation (the below sample based on 9 classes and MobileNetV2 transfer learning)

Comparison	MobileNetV2 mostly trainable parameter	MobileNetV2 with set the base parameter as non-trainable
Trainable Parameter	Total params: 4,265,353 Trainable params: 4,231,241 Non-trainable params: 34,112	Total params: 4,265,353 Trainable params: 2,007,369 Non-trainable params: 2,257,984
Time per epoch	Vary from 52s – 1141s	21s

Model Accuracy		
Confusion Matrix		
Classification Report	Best accuracy (on testing dataset): 99.79% precision recall f1-score support c0 0.9900 0.9900 0.9900 100 c1 1.0000 1.0000 1.0000 101 c2 1.0000 1.0000 1.0000 101 c3 1.0000 1.0000 1.0000 115 c4 1.0000 1.0000 1.0000 118 c5 1.0000 1.0000 1.0000 98 c6 1.0000 1.0000 1.0000 100 c7 1.0000 1.0000 1.0000 104 c9 0.9898 0.9898 0.9898 98 accuracy 0.9979 935 macro avg 0.9978 0.9978 0.9978 935 weighted avg 0.9979 0.9979 0.9979 935	Best accuracy (on testing dataset): 81.93% precision recall f1-score support c0 0.9118 0.9300 0.9208 100 c1 0.7864 0.8020 0.7941 101 c2 0.9510 0.9604 0.9557 101 c3 0.5345 0.8087 0.6436 115 c4 0.9098 0.9407 0.9250 118 c5 0.9038 0.9592 0.9307 98 c6 0.5625 0.0900 0.1552 100 c7 0.8729 0.9904 0.9279 104 c9 0.9043 0.8673 0.8854 98 accuracy 0.8193 935 macro avg 0.8152 0.8165 0.7932 935 weighted avg 0.8125 0.8193 0.7936 935
Additional training epoch	Not being executed as model had been optimal with 40 epochs	Added into 120 Epoch, observed 5% increase in Model accuracy against testing data 

CNN Model based on body posture and face:

We also experiment with combining body posture (side view) and drowsy / alert face (front view) into one model using MobileNetV2 transfer learning.

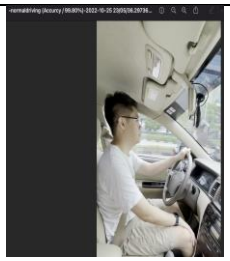
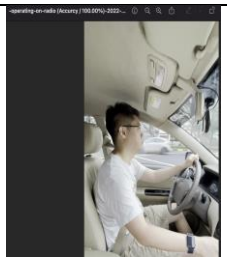
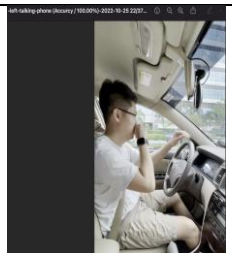
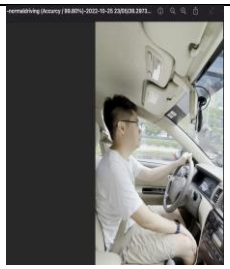
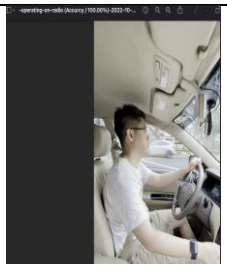
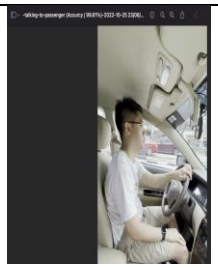
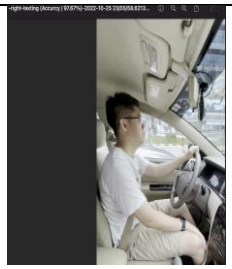


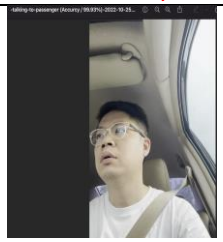
Best accuracy (on testing dataset): 94.89%

	precision	recall	f1-score	support
c0	0.9901	1.0000	0.9950	100
c1	1.0000	0.9901	0.9950	101
c10	0.7570	0.7364	0.7465	110
c11	0.7434	0.7636	0.7534	110
c2	1.0000	1.0000	1.0000	101
c3	1.0000	1.0000	1.0000	115
c4	1.0000	0.9746	0.9871	118
c5	1.0000	1.0000	1.0000	98
c6	1.0000	1.0000	1.0000	100
c7	1.0000	1.0000	1.0000	104
c9	0.9703	1.0000	0.9849	98
accuracy			0.9489	1155
macro avg	0.9510	0.9513	0.9511	1155
weighted avg	0.9490	0.9489	0.9489	1155

The result from the combined image shown that's the accuracy of 2 additional class is below the rest of classes.

Below is the comparison result in real life sample (the sample taken from video, and run through the program and cut some of the frames to put here as sample):

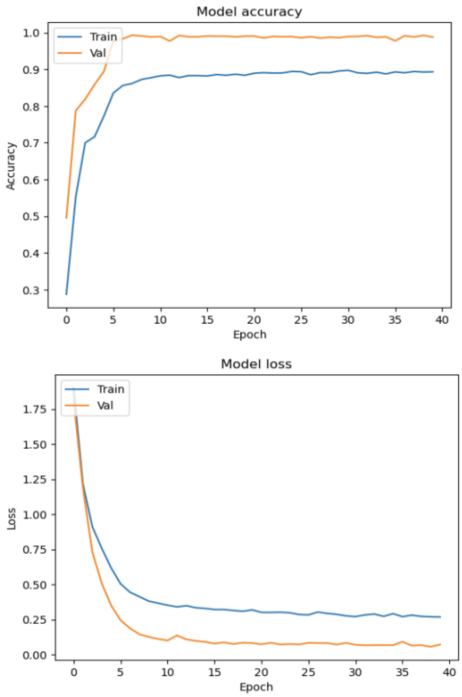
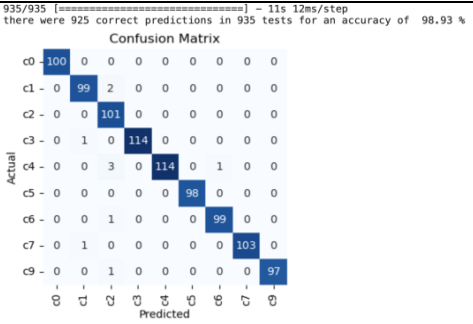
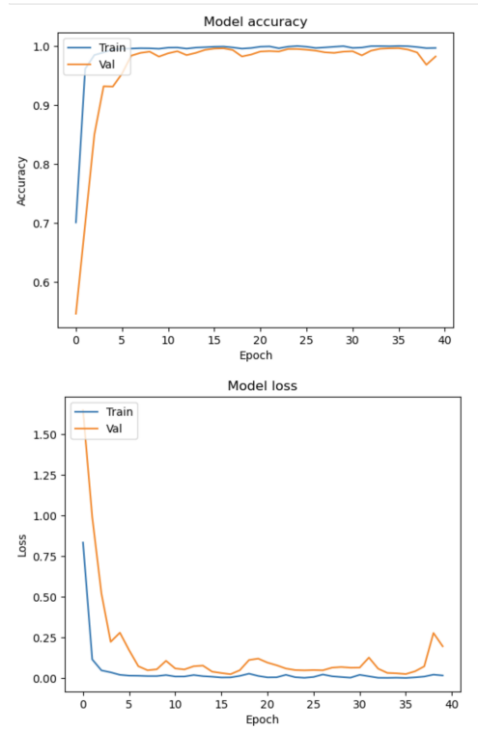
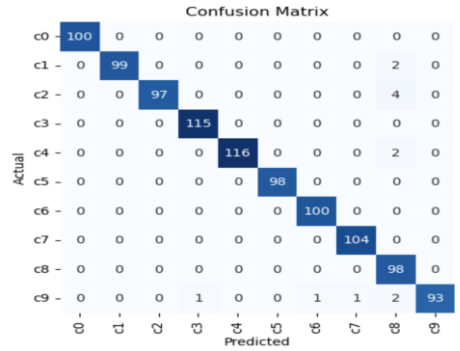
Model	Normal Driving	Operating on Radio	Talking to passenger	Others
Transfer learning MobileNetV2 – 9 classes	 99.91% confident	 100% confident	 96.34% confident	 100% confident Use left hand for phone (it's not fully accurate but user did some action in a moment)
Transfer learning MobileNetV2 – 11 classes side view	 99.80% confident	 100% confident		 97.67% confident (it's wrong classification)

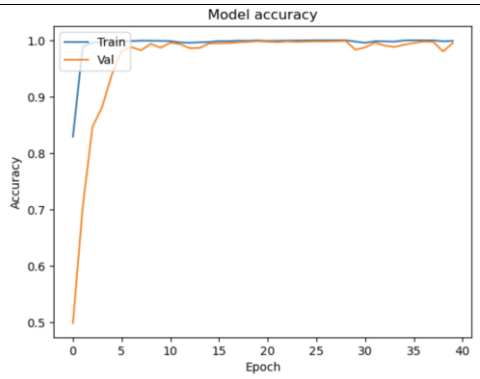
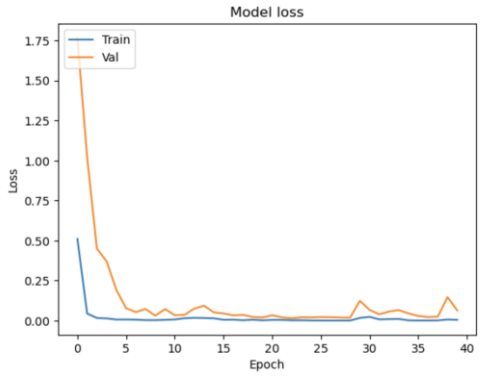
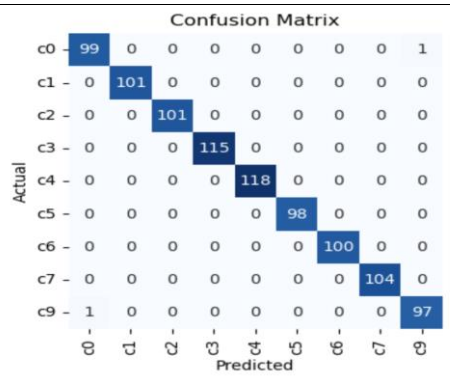
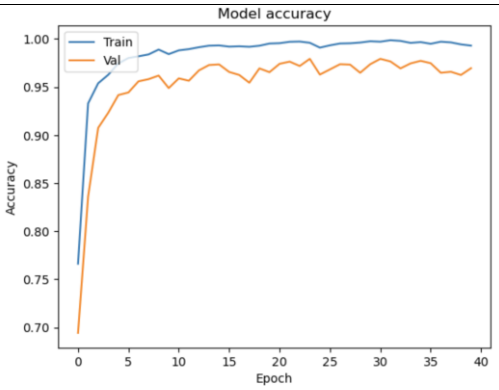
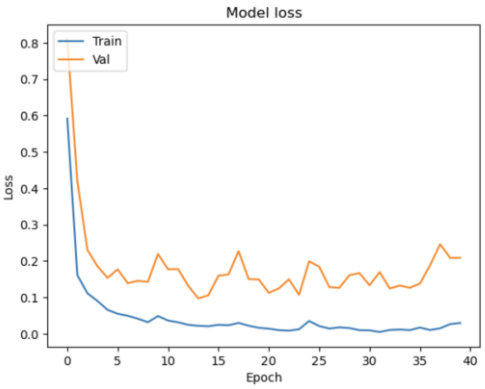
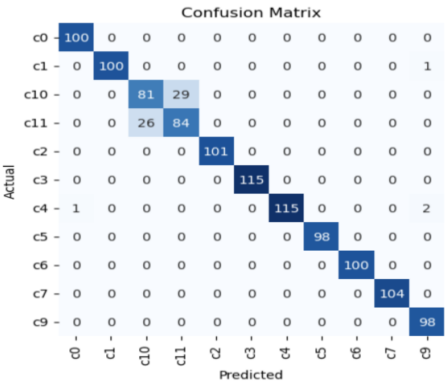
			99.61% confident (it's wrong classification)	
Transfer learning MobileNetV2 – 11 classes front view	 99.84% confident		 99.93% confident	

In summary, below are various model that being experiment:

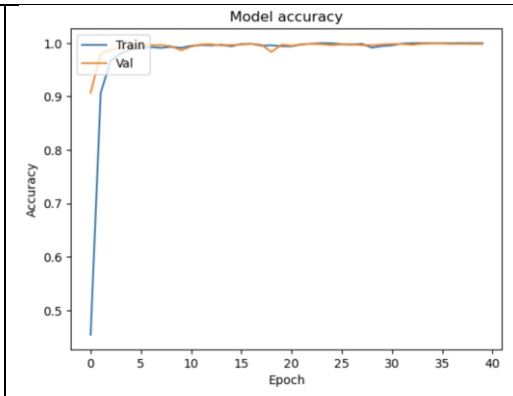
No	Model type	Explanation	Observation
1	Self-created model	This model created with dataset of 9 classes (side view) with: 2 convolutional layers 2 batch normalisations 2 depth wise convolutional Relu activation function	It take time and computation effort to build. With tuning and apply Depth wise convolutional the accuracy is improved but the model size is quite big (around 605MB).
2.	Transfer learning: MobileNetV2 with 10 classes	This model is transfer learning with body posture dataset with 10 classes (side view)	It's easily achieved higher accuracy with relatively small model size.
3.	Transfer learning: MobileNetV2 with 9 classes	This model is transfer learning with body posture dataset with 9 classes (side view), remove class of fixing hair / makeup as we want to avoid gender bias.	It's easily achieved higher accuracy with relatively small model size. More accurate in live test compare to 10 classes.
4.	Transfer learning: MobileNetV2 with 11 classes	This model is transfer learning with body posture dataset with 9 classes (side view), and add on alert driving and drowsy driving dataset (frontal view)	The model are rather confuse differentiated the 2 frontal view classes as describe in next table. Although the total accuracy quite high but in live test accuracy is lower than 9 classes.
5	Transfer learning: VGG16 with 9 classes	This model is transfer learning with body posture dataset with 9 classes (side view), remove class of fixing hair / makeup as we want to avoid gender bias.	The accuracy in training and test is very high (100%, as showed in next table) but the model size is large (3x the MobileNetV3) thus we deem this not optimal for edge device deployment.

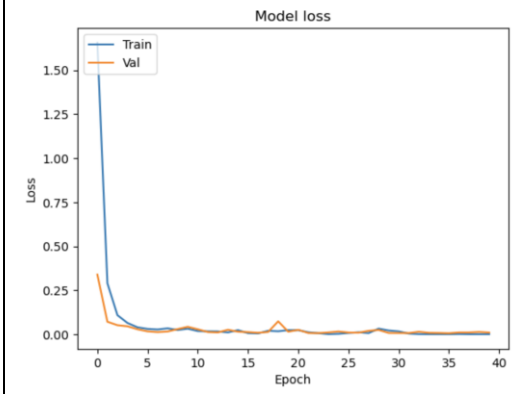
Comparison of results

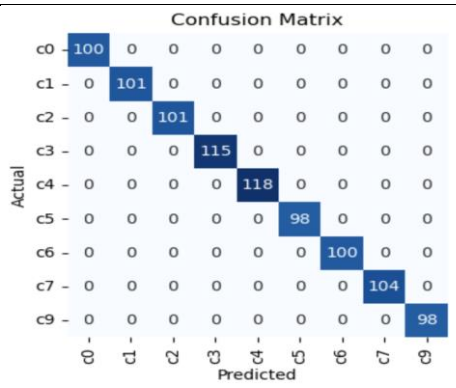
Model	Model Accuracy & Model Lost	Confusion matrix & Classification Report
Self created model – 9 classes		 Best accuracy (on testing dataset): 98.93% precision recall f1-score support c0 1.0000 1.0000 1.0000 100 c1 0.9802 0.9802 0.9802 101 c2 0.9352 1.0000 0.9665 101 c3 1.0000 0.9913 0.9956 115 c4 1.0000 0.9661 0.9828 118 c5 1.0000 1.0000 1.0000 98 c6 0.9900 0.9900 0.9900 100 c7 1.0000 0.9904 0.9952 104 c9 1.0000 0.9898 0.9949 98 accuracy 0.9893 935 macro avg 0.9895 0.9898 0.9895 935 weighted avg 0.9898 0.9893 0.9894 935
MobileNetV 2 – 10classes		 Best accuracy (on testing dataset): 98.74% precision recall f1-score support c0 1.0000 1.0000 1.0000 100 c1 1.0000 0.9802 0.9900 101 c2 1.0000 0.9604 0.9798 101 c3 0.9914 1.0000 0.9957 115 c4 1.0000 0.9831 0.9915 118 c5 1.0000 1.0000 1.0000 98 c6 0.9901 1.0000 0.9950 100 c7 0.9905 1.0000 0.9952 104 c8 0.9074 1.0000 0.9515 98 c9 1.0000 0.9490 0.9738 98 accuracy 0.9874 1033 macro avg 0.9879 0.9873 0.9872 1033 weighted avg 0.9883 0.9874 0.9875 1033

MobileNetV 2 – 9classes	 	 Best accuracy (on testing dataset): 99.79% precision recall f1-score support c0 0.9900 0.9900 0.9900 100 c1 1.0000 1.0000 1.0000 101 c2 1.0000 1.0000 1.0000 101 c3 1.0000 1.0000 1.0000 115 c4 1.0000 1.0000 1.0000 118 c5 1.0000 1.0000 1.0000 98 c6 1.0000 1.0000 1.0000 100 c7 1.0000 1.0000 1.0000 104 c9 0.9898 0.9898 0.9898 98 accuracy 0.9979 935 macro avg 0.9978 0.9978 0.9978 935 weighted avg 0.9979 0.9979 0.9979 935
MobileNetV 2 – 11classes	 	 Best accuracy (on testing dataset): 94.89% precision recall f1-score support c0 0.9901 1.0000 0.9950 100 c1 1.0000 0.9901 0.9950 101 c10 0.7570 0.7364 0.7465 110 c11 0.7434 0.7636 0.7534 110 c2 1.0000 1.0000 1.0000 101 c3 1.0000 1.0000 1.0000 115 c4 1.0000 0.9746 0.9871 118 c5 1.0000 1.0000 1.0000 98 c6 1.0000 1.0000 1.0000 100 c7 1.0000 1.0000 1.0000 104 c9 0.9703 1.0000 0.9849 98 accuracy 0.9489 1155 macro avg 0.9510 0.9513 0.9511 1155 weighted avg 0.9490 0.9489 0.9489 1155

VGG16 – 9 classes

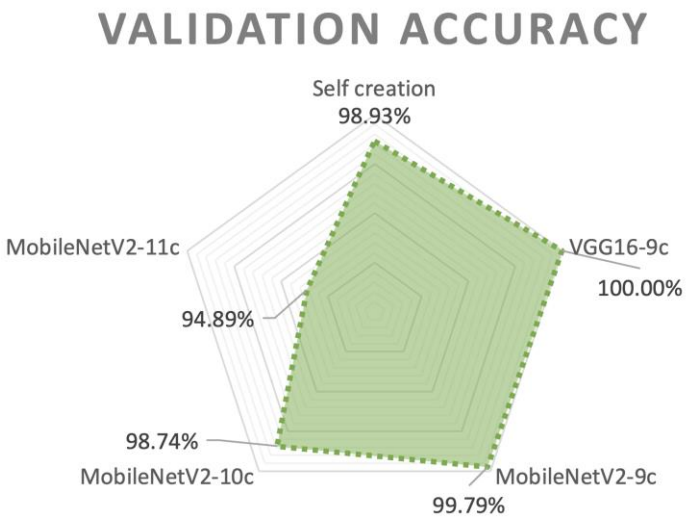






Best accuracy (on testing dataset): 100.00%

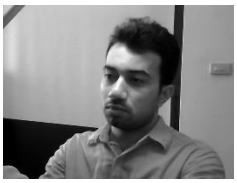
	precision	recall	f1-score	support
c0	1.0000	1.0000	1.0000	100
c1	1.0000	1.0000	1.0000	101
c2	1.0000	1.0000	1.0000	101
c3	1.0000	1.0000	1.0000	115
c4	1.0000	1.0000	1.0000	118
c5	1.0000	1.0000	1.0000	98
c6	1.0000	1.0000	1.0000	100
c7	1.0000	1.0000	1.0000	104
c9	1.0000	1.0000	1.0000	98
accuracy			1.0000	935
macro avg	1.0000	1.0000	1.0000	935
weighted avg	1.0000	1.0000	1.0000	935



All in all, MobileNetV2 Transfer learning with 9 classes is given the optimal result in training and validation as well as live test. It provides a relatively small size model after conversion to tflite (16MB). In term of accuracy on test dataset, mostly are perform relatively good.

APPROACH 3 – CNN+RNN MODEL

When the classification is performed on individual frames separately, sometimes it is hard to tell even to a human eye, whether the driver is drowsy or not. For example,



However, the same frame when put into the context, we can immediately tell, the driver is drowsy.

The same applies to machine learning models. If we train the model based on individual frames, the accuracy suffers in many scenarios. The labels of some of the images should even be questioned. That is why we also explored an alternative approach for the system: to feed a sequence of frames into the model so that the model understands what actions have happened across a time span.

DATA PREPARATION

The dataset used for this model is obtained from National Tsing Hua University (Ching-Hua Weng, 2016)

Data Source	Description	Type	Data Size (No. of Samples)	Data Size
National Tsing Hua University – Driver Drowsiness Detection Dataset	Dataset with 36 individuals in various simulated driving scenarios, including *normal driving, yawning, slow blink rate, falling asleep, etc.*, under day and night illumination conditions.	Videos	Training + Validation – 90 videos (1-1.5mins each, total 9.5hrs) Testing – 90 videos	± 7.45GB

Each video is about 100 seconds long and there is a mixture of drowsy sections and alert sections. For example, one of the videos has the following frames:

> <u>A total of 1852 frames. The “0” frames are alertness and “1” frames are drowsiness</u>
>
> 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
> 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
> 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
> 000000000000000000001111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111110000000000000000000000000000000
> 000000000000000000000000000000000000000000000000000000000000000000000000000000000011111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111110000000000000000000000000000000000000000000000000000000000001111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110000000000000000000000
> 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011111111111111111111111111
> 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

We must feed videos of the same type to the model for training. Therefore, the dataset cannot be used as-is for our project. We decided to make video clips of the same type (all “0” frames or “1” frames). The design is to build 10-second videos clips of the same type out of the given dataset. The assumption (human judgement) here is that a 10-second video is long enough to give good information to determine whether the driver is drowsy or not, while it is short enough to be processed with limited compute power.

Therefore, the training and testing data was produced by processing the videos obtained and made into 10-second clips. To do this, we developed a function based on OpenCV to read 10 seconds of frames (10s * 30 fps=300 frames) of the same label and save as a video file. Frame by frame, the

corresponding label in the txt file is checked before it is saved. This makes sure the video clip is of the same type (drowsy or non-drowsy). After preparation, we have the following set of videos for training and testing:

Class 0 – Alertness – 88 x 10-second videos

Class 1 – Drowsiness – 88 x 10-second videos

One consideration here is also about the eventual deployment. On an edge device, the number of frames that can be processed as one sequence. Based on the experiments with sample videos, we can use 20 frames as one sequence as a balance between sufficient information and the limited compute power. We think this works based on the nature of the type of videos we need to process. It is camera captures of drivers. This type of videos usually does not have drastic changes. When a driver becomes drowsy, it is also a gradual change process. So, it is not critical to capture very single frame. In fact, we can skip frames and when capturing and this allows us to capture 6 seconds of content just with 20 frames. This is achieved through capturing one frame in every 10 frames across 201 frames in a 30-fps video. We implemented this frame skipping logic in our deployed inference code.

MODEL BUILDING

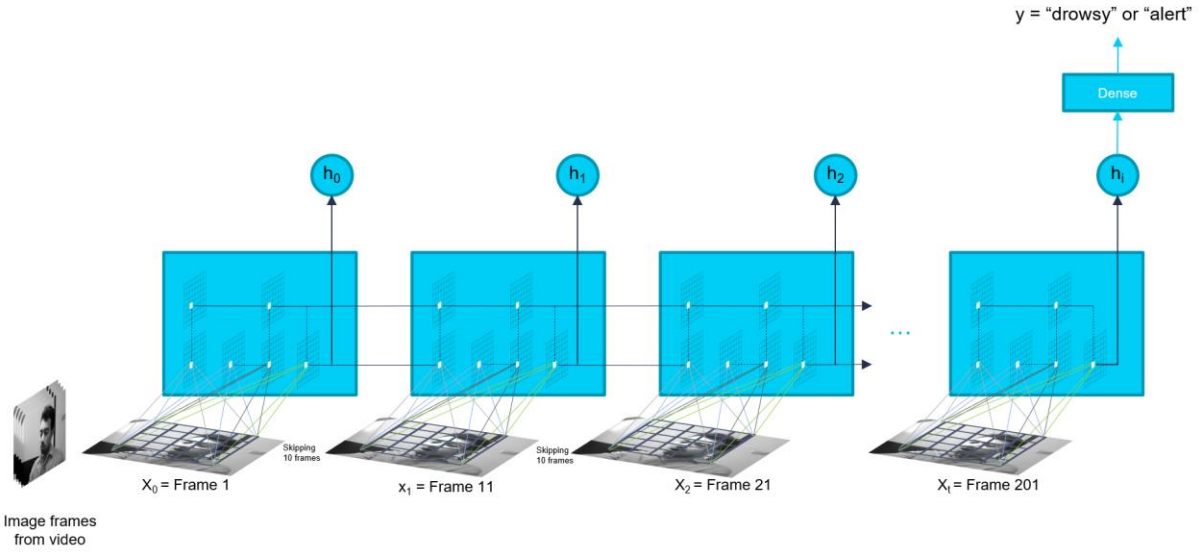
The idea of CNN+RNN is to use a sequence of frames as the input data so that the model understands what event is happening. Through research, we learned and implemented two different approaches to this. Initially, we started with Approach 1 which is ConvLSTM (Convolutional Long Short-Term Memory). However, we realised that the overall accuracy is low initially and it overfits. While we tried different approaches to optimise the same model, we also explored further and tried Approach 2 – LRCN (Long-term Recurrent Convolutional Network). LRCN produced better stability and there is much less sign of overfitting. It also trains faster.

Approach 1: ConvLSTM (Convolutional Long Short-Term Memory)

This is the first approach we tried. It is an existing Keras class (ConvLSTM2D layer, n.d.) that can be used. The structure of ConvLSTM network is essentially the same as a typical LSTM. The difference is that inside each cell, the matrix multiplication between input and weights is changed to convolutions.

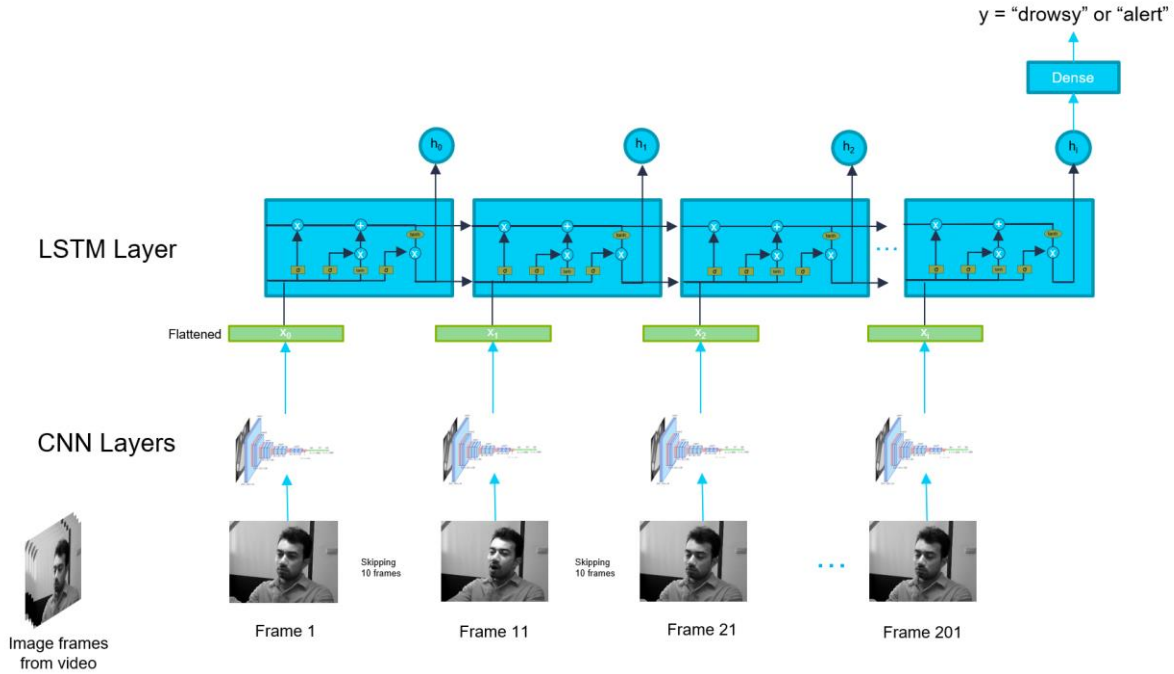
This way, it can take image inputs directly, extracting the features inside the LSTM cells and feeding them to the next time step to be combined with the convolution result of the next input frame. There is a research paper that provided a good grasp of the concept and technical details (Medel, 2016).

The diagram below was drawn based on our understanding of the ConvLSTM model and our implementation in this project.



Approach 2: LRCN (Long-term Recurrent Convolutional Network)

This is the second approach we implemented. Compared to the ConvLSTM approach, it requires the image to be processed first before feeding into the LSTM cell. Compared to ConvLSTM, it does perform less rounds of convolution computation as what is fed into the input of LSTM is the flattened vector output from the earlier convolutional layers. The diagram below illustrates the architecture of the model.



One key difference between ConvLSTM and LRCN in input is that the inputs (x_0, x_1, x_2...x_t) in ConvLSTM are picture data directly while the input in LRCN are vectors as the result of CNN layers.

Iterations and findings

In the initial model, we tried feeding the entire 10-second video (300) frames into the model. This turned out too big for the compute power available to us. The biggest computer we are using is the one provided by Google Colab pro+ subscription which provides:

- **RAM**: 90GB
- **GPU**: NVIDIA-SMI 460.32.03; Driver Version: 460.32.03; CUDA Version: 11.2; GPU Memory size: 40GB

Even with the compute power above, the model was not able to train as it crashes the machine. Therefore, we decided to reduce the number of frames per sequence. This consideration is not just for training. We also need to consider eventual deployment which is on an edge device that has much less compute power. To decide what number of frames is good. We took the following factors into consideration:

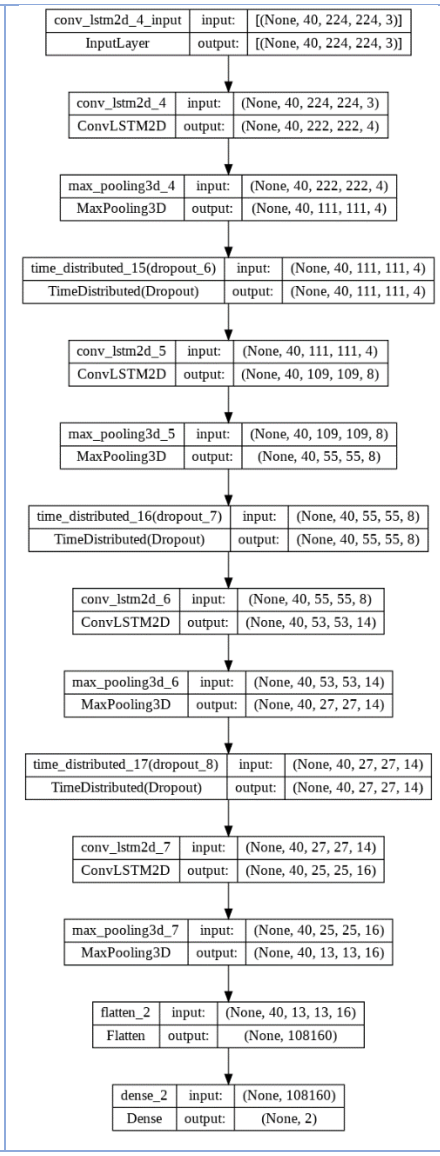
1. **Deploy-ability**
   After quick deployment test on Raspberry Pi. A TensorFlow model with 20 frames per sequence can run at near full CPU. 40 frames and above had the risk of device failure.
2. **Model performance**
   We compare how the number of frames per sequence impacts the model stability and accuracy. We noticed that 20 frames yields better accuracy than 40 frames across both approaches.

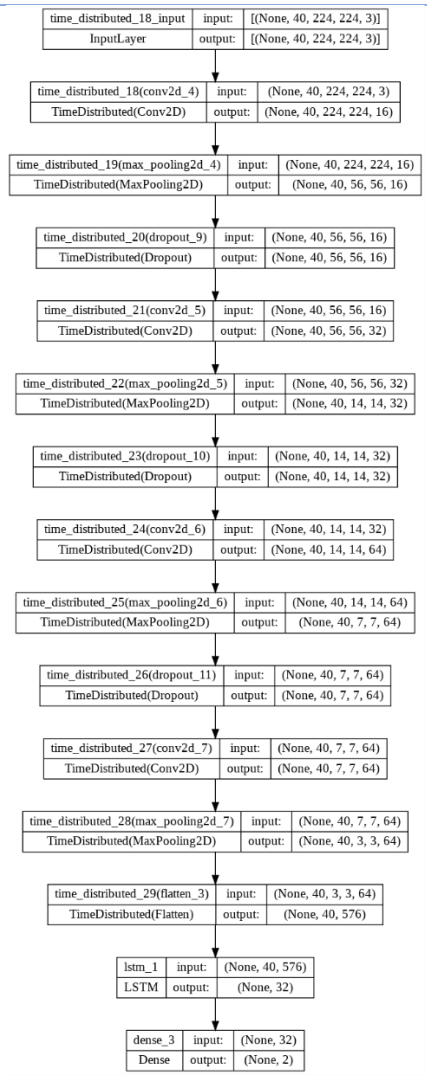
On top of the above, 20 frames per second also has a higher training speed. Therefore, we decide to keep the model at 20 frames per second.

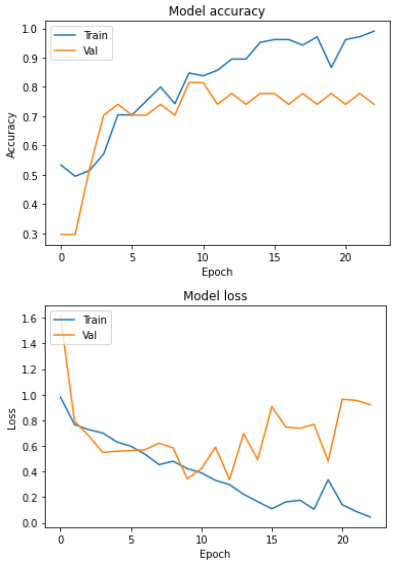
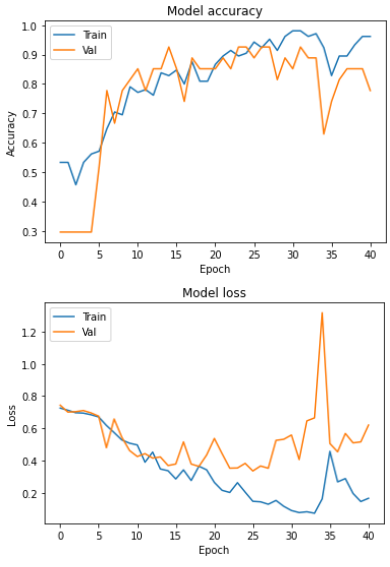
One more note is on the skipping of frames. On a video that is 30 frames per second, if we take 20 consecutive frames, it is less than 1 second of content. It does not contain sufficient information for inference. Each frame should look similar as well because in a car-driving setting, the changes are not drastic. Therefore, it is good to take the 20 frames out of 200 frames, skipping 10 frames at each step, this will capture 6-7 seconds of information and good enough to determine if the driver is drowsy or not.

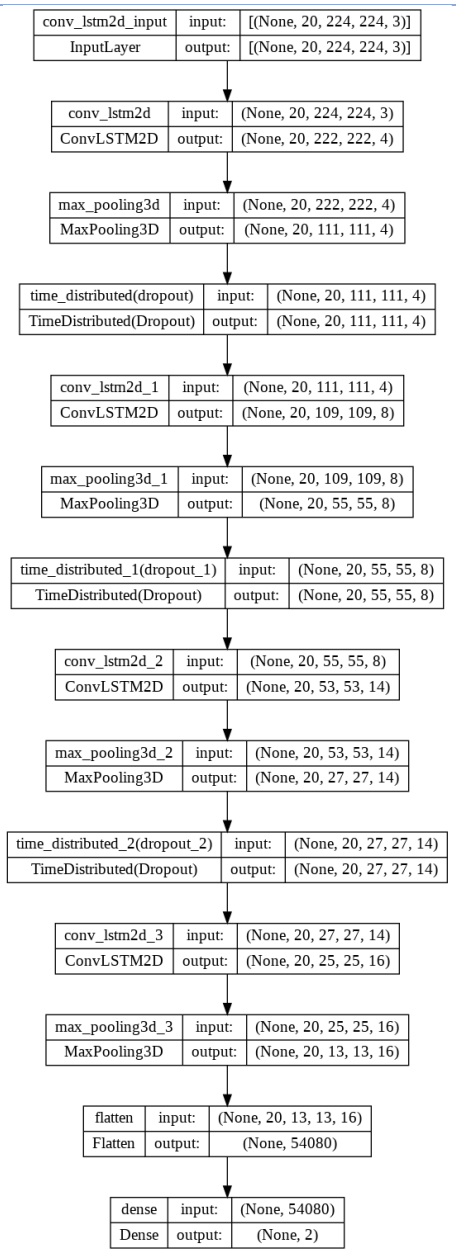
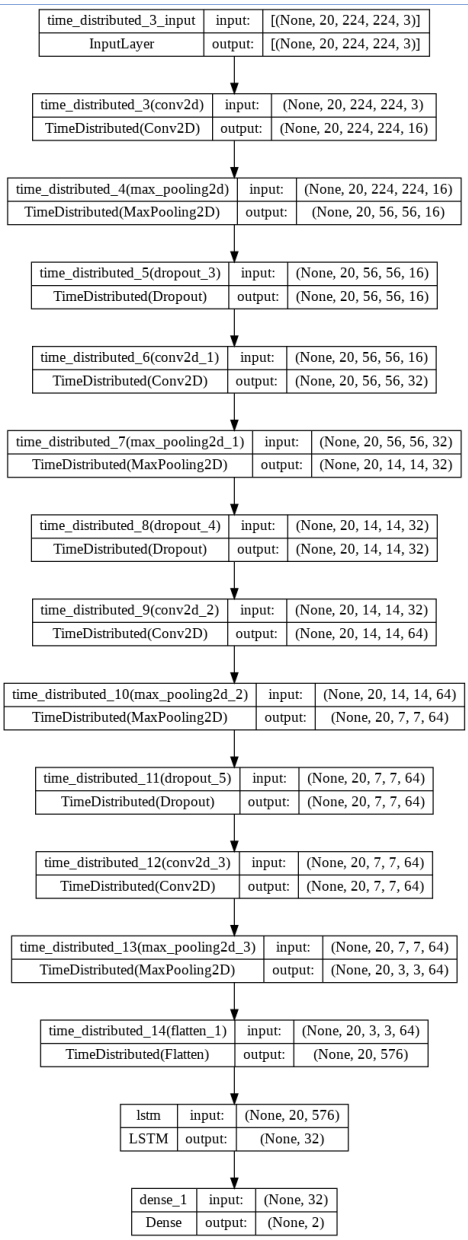
Version	**V1** **40 frames per sequence**	
Model	ConvLSTM	LRCN

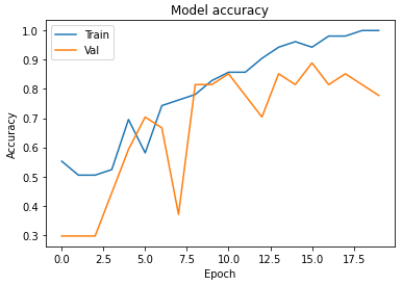
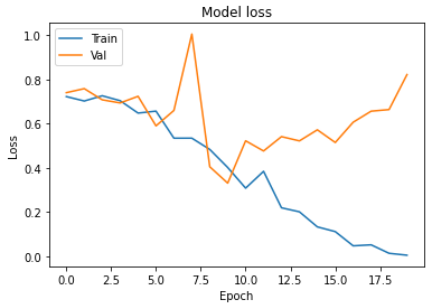
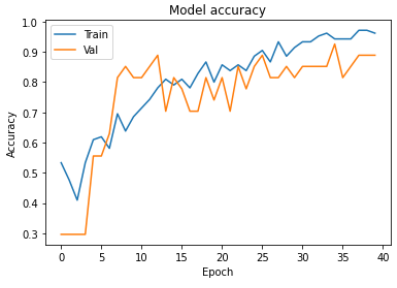
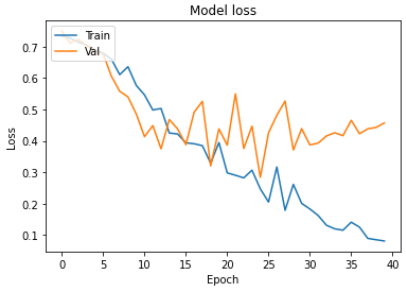
Model Structure





No. of Params	Layer (type) Output Shape Param # conv_lstm2d_4 (ConvLSTM2D) (None, 40, 222, 222, 4) 1024 max_pooling3d_4 (MaxPooling3D) (None, 40, 111, 111, 4) 0 time_distributed_15 (TimeDistributed) (None, 40, 111, 111, 4) 0 conv_lstm2d_5 (ConvLSTM2D) (None, 40, 109, 109, 8) 3488 max_pooling3d_5 (MaxPooling3D) (None, 40, 55, 55, 8) 0 time_distributed_16 (TimeDistributed) (None, 40, 55, 55, 8) 0 conv_lstm2d_6 (ConvLSTM2D) (None, 40, 53, 53, 14) 11144 max_pooling3d_6 (MaxPooling3D) (None, 40, 27, 27, 14) 0 time_distributed_17 (TimeDistributed) (None, 40, 27, 27, 14) 0 conv_lstm2d_7 (ConvLSTM2D) (None, 40, 25, 25, 16) 17344 max_pooling3d_7 (MaxPooling3D) (None, 40, 13, 13, 16) 0 flatten_2 (Flatten) (None, 108160) 0 dense_2 (Dense) (None, 2) 216322 Total params: 249,322 Trainable params: 249,322 Non-trainable params: 0 Model Created Successfully!	Layer (type) Output Shape Param # time_distributed_18 (TimeDistributed) (None, 40, 224, 224, 16) 448 time_distributed_19 (TimeDistributed) (None, 40, 56, 56, 16) 0 time_distributed_20 (TimeDistributed) (None, 40, 56, 56, 16) 0 time_distributed_21 (TimeDistributed) (None, 40, 56, 56, 32) 4640 time_distributed_22 (TimeDistributed) (None, 40, 14, 14, 32) 0 time_distributed_23 (TimeDistributed) (None, 40, 14, 14, 32) 0 time_distributed_24 (TimeDistributed) (None, 40, 14, 14, 64) 18496 time_distributed_25 (TimeDistributed) (None, 40, 7, 7, 64) 0 time_distributed_26 (TimeDistributed) (None, 40, 7, 7, 64) 0 time_distributed_27 (TimeDistributed) (None, 40, 7, 7, 64) 36928 time_distributed_28 (TimeDistributed) (None, 40, 3, 3, 64) 0 time_distributed_29 (TimeDistributed) (None, 40, 576) 0 lstm_1 (LSTM) (None, 32) 77952 dense_3 (Dense) (None, 2) 66 Total params: 138,530 Trainable params: 138,530 Non-trainable params: 0 Model Created Successfully!
Train Speed	884ms/step at batch_size =4	52ms/step at batch_size =4
Model Accuracy	Accuracy: 0.7045454545454546 Confusion Matrix: [[18 2] [11 13]] classification_report: precision recall f1-score support 0 0.62 0.90 0.73 20 1 0.87 0.54 0.67 24 accuracy 0.70 44 macro avg 0.74 0.72 0.70 44 weighted avg 0.75 0.70 0.70 44	Accuracy: 0.8409090909090909 Confusion Matrix: [[15 5] [2 22]] classification_report: precision recall f1-score support 0 0.88 0.75 0.81 20 1 0.81 0.92 0.86 24 accuracy 0.84 44 macro avg 0.85 0.83 0.84 44 weighted avg 0.85 0.84 0.84 44
Training History Plots		

Version	V2 20 frames per sequence	
Model	ConvLSTM	LRCN
Model Structure		

No. of Params	Layer (type) Output Shape Param # conv_lstm2d (ConvLSTM2D) (None, 20, 222, 222, 4) 1024 max_pooling3d (MaxPooling3D (None, 20, 111, 111, 4) 0 time_distributed (TimeDistr (None, 20, 111, 111, 4) 0 conv_lstm2d_1 (ConvLSTM2D) (None, 20, 109, 109, 8) 3488 max_pooling3d_1 (MaxPooling (None, 20, 55, 55, 8) 0 time_distributed_1 (TimeDis (None, 20, 55, 55, 8) 0 conv_lstm2d_2 (ConvLSTM2D) (None, 20, 53, 53, 14) 11144 max_pooling3d_2 (MaxPooling (None, 20, 27, 27, 14) 0 time_distributed_2 (TimeDis (None, 20, 27, 27, 14) 0 conv_lstm2d_3 (ConvLSTM2D) (None, 20, 25, 25, 16) 17344 max_pooling3d_3 (MaxPooling (None, 20, 13, 13, 16) 0 flatten (Flatten) (None, 54080) 0 dense (Dense) (None, 2) 108162 Total params: 141,162 Trainable params: 141,162 Non-trainable params: 0 Model Created Successfully!	Layer (type) Output Shape Param # time_distributed_3 (TimeDis (None, 20, 224, 224, 16) 448 time_distributed_4 (TimeDis (None, 20, 56, 56, 16) 0 time_distributed_5 (TimeDis (None, 20, 56, 56, 16) 0 time_distributed_6 (TimeDis (None, 20, 56, 56, 32) 4640 time_distributed_7 (TimeDis (None, 20, 14, 14, 32) 0 time_distributed_8 (TimeDis (None, 20, 14, 14, 32) 0 time_distributed_9 (TimeDis (None, 20, 14, 14, 64) 18496 time_distributed_10 (TimeDi (None, 20, 7, 7, 64) 0 time_distributed_11 (TimeDi (None, 20, 7, 7, 64) 0 time_distributed_12 (TimeDi (None, 20, 7, 7, 64) 36928 time_distributed_13 (TimeDi (None, 20, 3, 3, 64) 0 time_distributed_14 (TimeDi (None, 20, 576) 0 lstm (LSTM) (None, 32) 77952 dense_1 (Dense) (None, 2) 66 Total params: 138,530 Trainable params: 138,530 Non-trainable params: 0 Model Created Successfully!
Train Speed	438ms/step at batch_size =4	29ms/step at batch_size =4
Model Accuracy	Accuracy: 0.9318181818181818 Confusion Matrix: [[18 2] [1 23]] classification_report: precision recall f1-score support 0 0.95 0.90 0.92 20 1 0.92 0.96 0.94 24 accuracy 0.93 44 macro avg 0.93 0.93 0.93 44 weighted avg 0.93 0.93 0.93 44	Accuracy: 0.8409090909090909 Confusion Matrix: [[15 5] [2 22]] classification_report: precision recall f1-score support 0 0.88 0.75 0.81 20 1 0.81 0.92 0.86 24 accuracy 0.84 44 macro avg 0.85 0.83 0.84 44 weighted avg 0.85 0.84 0.84 44
Training History Plots	 	 

In addition, through running iterations of training with both models, we noticed that ConvLSTM has overall better accuracy at 20 frames per sequence, but LRCN also has advantages.

1. **ConvLSTM has better test accuracy**. In V1-40-frame, LRCN has a higher accuracy than ConvLSTM (ConvLSTM 70% vs LRCN 84%). In V2-20-frame training, LRCN has a lower accuracy (ConvLSTM 93% vs LRCN 84%). This shows that LRCN is more stable and ConvLSTM's accuracy varies based on data. We think this difference comes from the complexity of the ConvLSTM model. Convolution is done at the input, forget and output gates. So longer sequence make give the model too much information to process so that it negatively impacted the performance.
2. **ConvLSTM also has higher tendency to overfit**. Based on the training history, ConvLSTM has a higher tendency of overfitting. We also think this comes from the model complexity of ConvLSTM.
3. **LRCN has a higher training and inference speed.** For a batch_size of 4, LRCN trains at 52 ms/step while ConvLSTM is at 884 ms/step.

To optimise the model, to following has also been done during the training process.

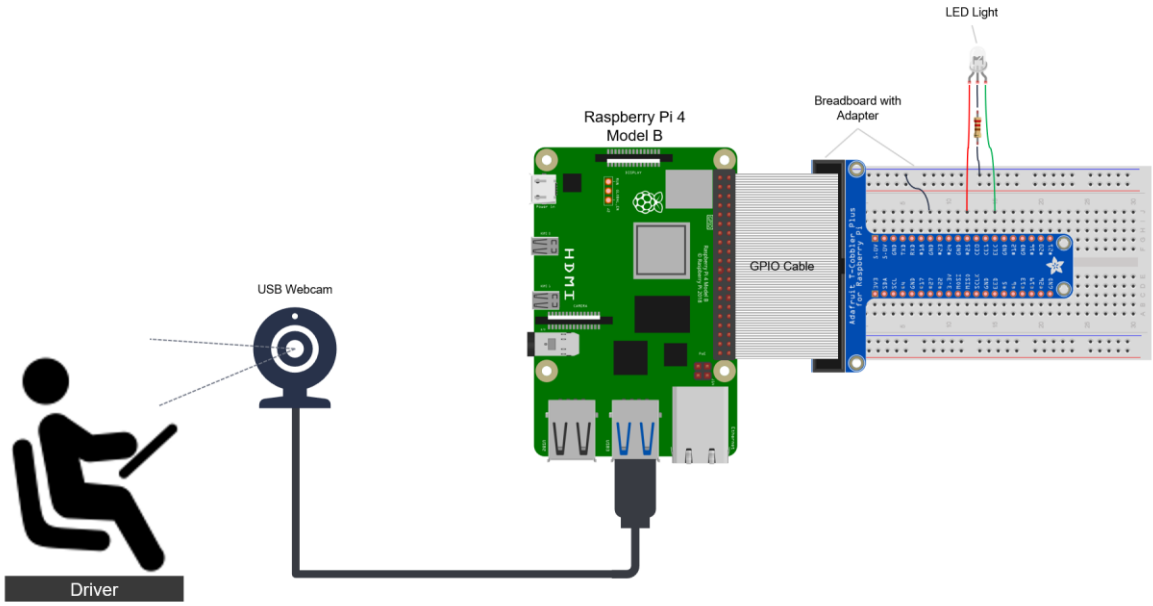
1. When preparing data, we made sure the data between the classes are balanced. There are the same number of videos in the dataset for each class.
2. Dropout layers were added to the models. This helped reduce overfit.

DEPLOYMENT TO EDGE DEVICE

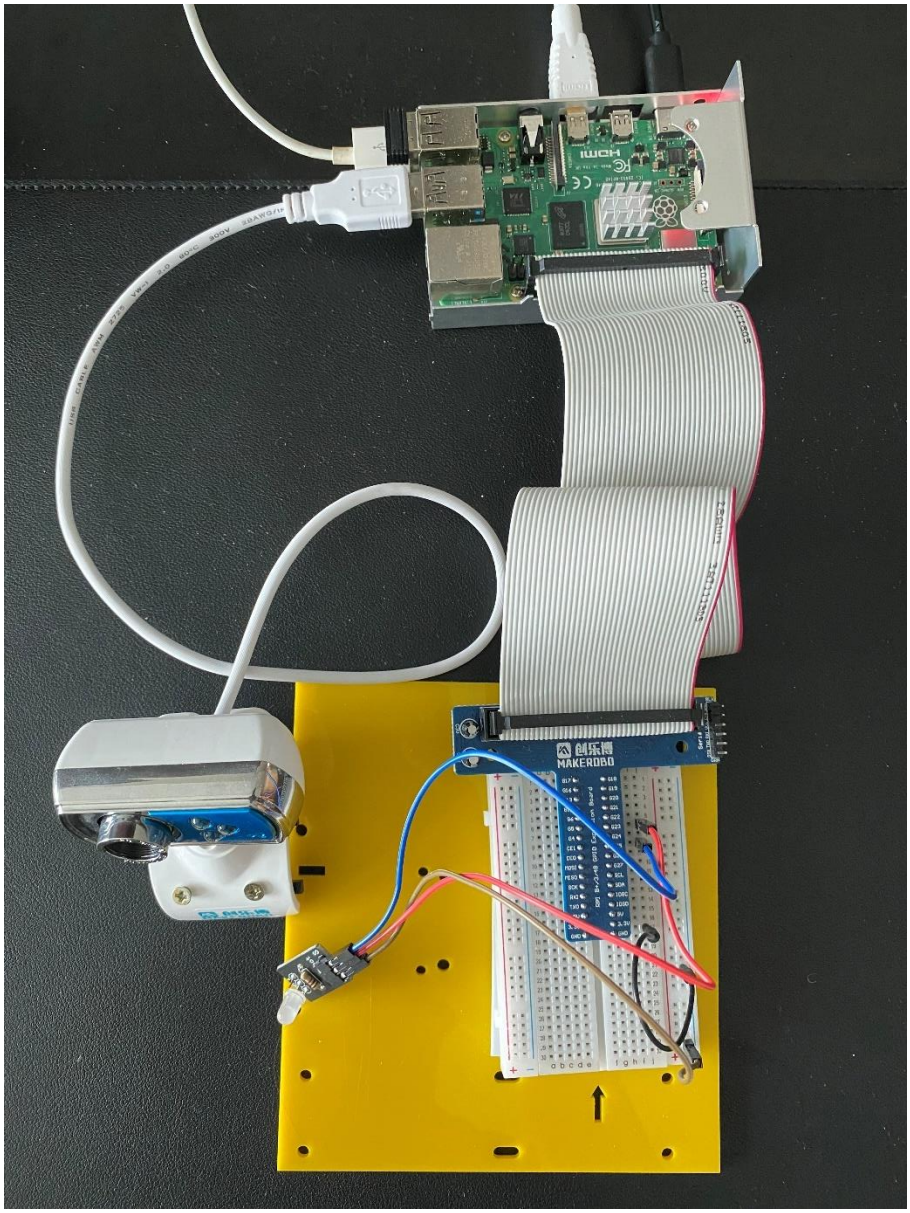
The model was trained with powerful machines with premium GPUs running. It is never realistic, or necessary, to use the same machines for inferences. Our project aims to build models that can be deployed on Raspberry Pi. There are two aspects to implement: hardware setup and model deployment.

Hardware Setup

The diagram below documents what is implemented in hardware, from camera for live capturing the driver's images to the LED light extended from the GPIO panel Raspberry Pi.



The actual hardware setup:



When the program runs, the driver's state is monitored actively. When it is detected that the driver is in a "drowsy" state, the red color LED light blinks 3 times.

Model Deployment

Model is initially created with the TensorFlow model and later converted into TensorFlow Lite model using the attached converter to be deployed into Raspberry Pi.

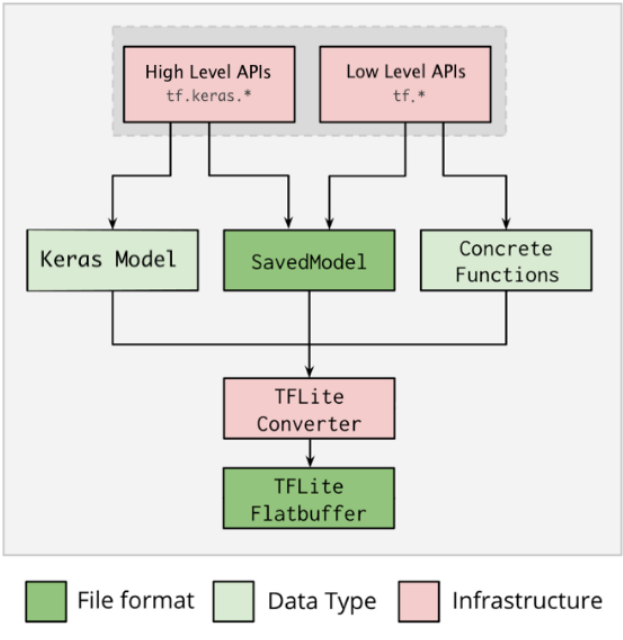


Figure 1. TensorFlow Lite conversion workflow

[TensorFlow (2022) Convert TensorFlow models [Image]. Convert TensorFlow models | TensorFlow Lite]

The conversion of CNN model into tensorflow lite is straightforward and during deployment in edge device, we could use tflite_runtime Interpreter to access the model. However this is not the case with RNN model, certain parameter is required to be activated during conversion and additional tensorflow component need to be installed. The ones in comments below are the options to turn on for RNN models.

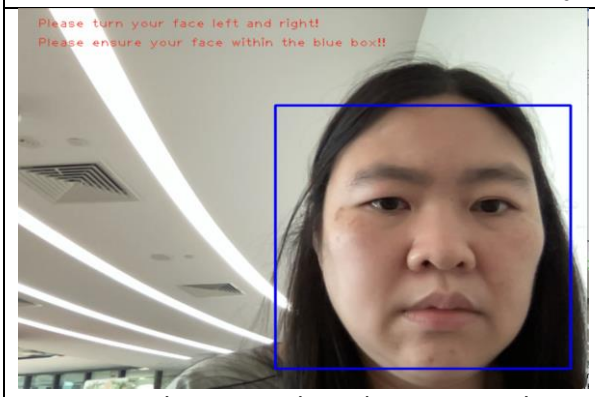
```
#converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
#                                       tf.lite.OpsSet.SELECT_TF_OPS]
#converter._experimental_lower_tensor_list_ops = False
```

The conversion to tensorflow lite will reduce the size of model roughly around 70% from initial size (i.e 51MB model in tensorflow will become 16MB model in tensorflow lite)

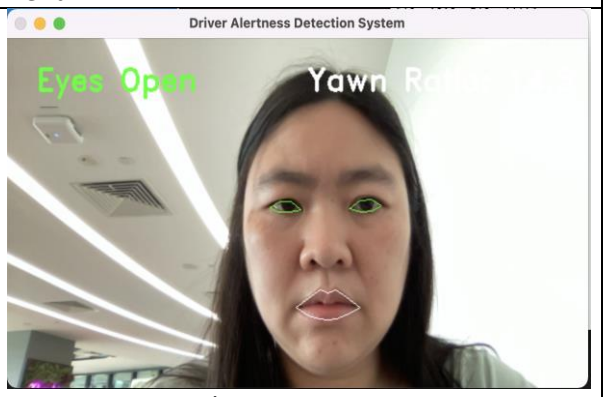
User Interface

The end product is a working model that is deployed on Raspberry Pi. To make it easier to demonstrate our product when Raspberry Pi is not available, we have also developed a UI that can be run on a laptop for the user to interact with our models. In this product demonstration we have 4 menus:

Main Menu

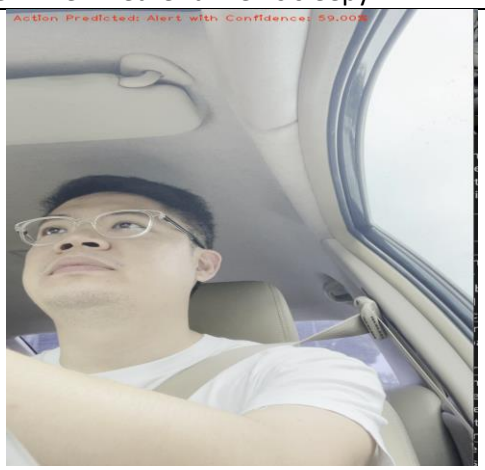


Register Me! – Upon selected Register Me! Camera will opne and system will recognize a face based on object detection, capture the images and encode this to pickle file to be used for other menus.



Front Camera Sleepiness Monitoring – Upon selected this options will monitor whether the driver is registered user and will use the recognize face to detect eyes and mouth to determine whether driver is sleepy.

Side Camera Alertness Monitoring - Upon clicking camera will open and system will get frame images and pass to the model to get classification. If more x classification not consider as normal driving system will inform driver to focus.



Front Camera Alertness Monitoring - Upon clicking camera will open and system will get x frame of images and pass to the model to get classification.

No	Menu	Features
1.	Register Me!	This menu will open camera and system will capture cv2 cascade classifier to detect face and camera will be opened for 100ms or until ESC key being called. During camera open, system will capture and

		save and the face images. The purpose of this is to keep driver face for detection purposes.
2.	Front Camera Sleepiness Monitoring	The main intention of this menu is to identify whether the driver sleepy based on the facial landmark model. System will identify whether the face is the registered driver and once confirmed it will then start to identify whether the driver is sleepy.
3.	Side Camera Alertness Monitoring	The intention of this menu is to identify whether driver is distracted due to performing some other activities other than driving. The team recognize that some driver might not locate their phone or device in front of them. So, the team would like to provide options alertness detection from side view.
4.	Front Camera Alertness Monitoring	While sleepiness detection is recognizing certain facial landmark, this alertness monitoring is being introduce as ro identify drivers that feel tired required other aspect in overall face expression to recognize whether drivers are alert.

Source guide

No	Folders	Files	Remark
1.	cnn_result	NA	This folder is used to keep images that being slice from video to test accuracy of CNN model.
2.	face_record	NA	This folder is used to keep driver photo from Register Me button.
3.	models		This folder is used to keep various different model and it's supporting files. RNN Models: ConvLSTM_26102022.h5 LRCN_model_24102022.h5 Face Landmark Model: best_model_test.tflite haarcascade_frontalface_default.xml shape_predictor_68_face_landmarks.dat CNN body posture + face models: MobileNetV2 9 classes – 200 epoch: tl_driver_distraction_model_02.tflite label.txt Self create model 9 classes – 100 epoch: sc_32pxl_09c.tflite label_sc_32pxl_09c.txt MobileNetV2 9 classes (32 pixel)– 100 epoch: MobileNetV2_32pxl_09c.tflite label_MobileNetV2_32pxl_09c.txt MobileNetV2 9 classes (224 pixel)– 40 epoch: trxlrn_MobileNetV2_09c.tflite label_trxlrn_MobileNetV2_09c.txt

			MobileNetV2 10 classes (224 pixel)– 40 epoch: trxlrn_MobileNetV2_10c.tflite label_trxlrn_MobileNetV2_10c.txt MobileNetV2 11 classes (224 pixel)– 40 epoch: trxlrn_MobileNetV2_11c.tflite label_trxlrn_MobileNetV2_11c.txt VGG16 9 classes (224 pixel)– 40 epoch: trxlrn_VGG16_09c.tflite
4.	sample_videos	NA	This folder is used to keep videos for testing the model.
5.	sounds		This folder is used to keep all sound that being used as alert.
6.	ui		This folder is used to keep all the source code. To test the system with various model and environment, we keep one constant.py file to manage all the difference testing. 
7.	Raspberry Pi		This folder keeps the source code deployed on Raspberry Pi.

PROJECT CONCLUSIONS: FINDINGS & RECOMMENDATION

COMPARING THE APPROACHES

The goal of all the models is to be able to detect that the driver is not focusing on driving. After building and tuning many different approaches separately, we review them together and compare them side by side.

Model	Pros	Cons
Approach 1, CNN for Facial Landmarks for Drowsiness.	Easy to train and achieve high accuracy.	It is per image classification. Needs additional logic to avoid classifying blinking (eyes closed) images as sleeping. (We have implemented it in our solution).

Approach 2, CNN for face and body postures (up to 11 classes)	It can classify not just drowsy drivers, but up to 10 different behaviours that are not normal driving. With transfer learning, it can achieve close to 100% accuracy in some cases.	It is hard to achieve high accuracy when trained from scratch. Transfer learning with pre-trained models is necessary to achieve high accuracy. It takes significantly longer time for training to achieve high accuracy. It required opening all layers in the pre-trained models and retrain the parameters.
Approach 3, CNN+RNN for Drowsiness	For a model that is trained from scratch (without any transfer learning), it is relatively easy to achieve higher accuracy, especially when compared with approach 2.	It does require more compute resources during inference as it processes sequences of images instead of individual ones.

We also have the following overall findings:

- The models with higher accuracy without transfer learning: Approach 1 and Approach 3. Both were able to achieve a testing accuracy of 90% by training the models from scratch.
    - For approach 1 with facial landmarks, this is because the task itself is focused. It detects the mouth and eyes and calculates the open ratio. High accuracy is achievable with good quality of data and model tuning. Our implementation has proven that.
    - For approach 3, it takes into consideration not just individual frames but a sequence of frames. This is good for detecting actions. Especially for subtle differences between drowsy and non-drowsy faces.
- Transfer learning is not always the silver bullet. In some implementation, it does not yield good performance. This was especially obvious in our Approach 1 implementation.
- Transfer learning becomes the more powerful if we also have relatively large compute power and able retrain the weights of more layers (instead of locking most layers untrainable).
- Data balance matters. The point has two aspects:
    - The data size and quantity across different classes must be as close as possible so that the model performs well across all classes.
    - The *distance* across different classes should be balanced as well. One of the models for Approach 2, we included two additional classed based on the person's face to the model, together with other 9 classes based on body postures. We can see in the test result that misclassifications were all between the 2 face-based classes. This shows that the models are not performing well on intricate differences between face features while it performs very well on the body postures. We have to say that the distance among the 9 body postures-based classes is larger than that between the two face feature based classes, thus the model performs better on the 9 body posture classes than the 2 face feature models.

PROJECT CONCLUSION

After more than two months of effort, we have successfully finished the project meeting the objectives of the project design. More importantly, we have had the opportunity to apply what we have learned in the courses with actual complete implementation from scratch. In this semester, we learned how software system can provide intelligent capabilities through intelligent pattern recognition and sense

making. In the group project, we were able to implement a system that monitors the driver's state through processing live stream of video content from camera and send out alerts when needed to help the driver stay alert, reducing the chances of accident resulted from sleepy or distracted drivers.

To recap the success measurement for this project, the end product should be **deployed on an edge device**, **only monitoring the registered driver** and **issuing alerts in-time when risk (driver drowsy or distracted) is detected**. We have met all the criteria.

- **Deployment**. We have deployed our models, built in different approaches, on Raspberry Pi 4 model B.
- **Identifying the driver**. Through the face identification function developed, the system is able to detect if the driver is in the frames captured live through camera. It will only proceed with driver state detection if the registered driver is in.
- **Issuing real-time alerts**. In our system deployed on Raspberry Pi, the alert is sent through the connected LED light. The red light will be flashing when the driver is detected to be drowsy or distracted.

Most importantly, we gained practical knowledge and experience in the following areas:

- **Data Preparation**
  Although our dataset was already pre-processed and even labelled when we got access, some dataset cannot be used directly. For example, the video dataset from Taiwan Tsinghua University had frame by frame labelling, but each video is a mixture of drowsy and non-drowsy frames. We need to further make clips of videos that only contains one type of state. When feeding the data to the model, we also need to decide then number of frames to feed each time. Such decision was made after experimenting with different numbers.
- **Model optimisation**
  To make the model perform well, we also applied the techniques learned in class:
  - **Adjusting model architecture and hyperparameter tuning:**
    For example, adding dropout layers to reduce overfit. Batch Normalization were also used for model stability. For CNN-RNN, we tried both ConvLSTM and LRCN and compared them. Hyper-parameter tuning also played an important part. After trying different sizes, we were aligned that all images used across all approaches and models to be resized to 224x224. In each of the models that were trained, we also tried with different learning rate, and activation functions.

  - **Transfer learning leveraging prebuilt models (with care):**
    This is to compare with the models we built and trained from scratch. We also tried locking and opening different layers untrainable to compare the difference in training time and accuracy. This is valuable experience as in our real-world implementation in the future, we believe transfer learning is an important part. Now through this project, we have a better understanding of when and how to use it. We also got ourselves familiarized with some of the popular models that are good for deployment on mobile devices.

  - **Optimisation for edge device deployment**:
    Converting the models trained on powerful machines into ones that can be run efficiently on small devices was an important part for the model to provide value. In this project, we experimented with different options in converting TensorFlow models to tflite ones.

- **Deployment**
  - **Hardware for deployment:**

Not all team members were familiar with Raspberry Pi before the project started. Through this project, we all got a good understanding of how Raspberry Pi stands out as a small computer for experiments, not just for its size and portability, but also for the GPIO panel that opens up many possibilities. That is why in this project, we were able to use a LED light connected through a breadboard.

- **Bundling a deployable package:**
  Through this learning we realised the importance of packaging the deployable software into bundle, as different OS version / difference camera type will have a big impact on how we could deploy our package.

We take this as an essential extension to the workshops in the individual lessons which focused on specific knowledge points. With the implementation of the project, we went from problem selection, ideation, design, development, testing to deployment. This end-to-end go-to-market process has enabled us to take our knowledge learned in class to the next level and further prepared us to meet the needs in real world scenarios. We believe that with the practical project implementation on top of the courses in each of the graduate certificates in our MTech IS program, we will be well-positioned to solve real problems with complete ready-to-use solutions. We look forward to the learnings of next semesters.

FUTURE IMPROVEMENTS

This implementation can be regarded as an initial launch for the product. Given more time, the following areas can be further enhanced in the future releases.

- **Data**
  - **Collection**:
    At the moment the dataset that is being used has images that are either from very align frontal face (eye level) or from the side, upper corner. This creates some challenges as driver will likely put phone sideway, upper or lower from face. So, the data collection will need to be from multiple angles so it would be able to cater for various location driver put their phone.

- **Model optimisation**
  - There is other various model optimisation that we can apply to improve model result especially when the data is available, e.g. taking concurrent measurement from side view and frontal view and use functional API to get one result.

- **Deployment**
  - **Expand device for deployment**
    As the measurement of driver alertness could be from multiple angles, there are opportunities to expand monitoring to other device as well, multiple sensors multiple measurements.

  - **Integration with real mobile app**
    There are some adjustments to be made to allow the model work with a mobile device. Especially on how we should allow the model to work effectively and at the same time we still be able to conserve the mobile device battery.

  - **Integration with GPS**
    At the moment, there is no differentiation whether currently the car is moving or not. This is a feature that good to be added.

APPENDIX A: USER MANUAL

Please find the User Manual in this folder on our Github repo: https://github.com/SionsML/PRS-PM-2022-07-02-ISY5002-GROUP_7/tree/master/Project%20Report

BIBLIOGRAPHY

Ching-Hua Weng, Y.-H. L.-H. (Nov, 2016). *Driver Drowsiness Detection Dataset*. Retrieved from National Tsing Hua University: http://cv.cs.nthu.edu.tw/php/callforpaper/datasets/DDD/

Computer Vision Lab, National Tsing Hua University. (November, 2016). *Driver Drowsiness Detection Dataset*. Retrieved from National Tsing Hua University website: http://cv.cs.nthu.edu.tw/php/callforpaper/datasets/DDD/

ConvLSTM2D layer. (n.d.). Retrieved from Keras API reference: https://keras.io/api/layers/recurrent_layers/conv_lstm2d/

Dexter, O. (6 June, 2018). *Drowsiness Monitoring System Using OpenCV And Tkinter Python 3*. Retrieved from https://www.youtube.com/watch?v=GWkcRj7GuYk

Evlanova, A. (27 April, 2021). Retrieved from valuechampion: https://www.valuechampion.sg/probability-car-accident

Geitgey, A. (10 June, 2022). *face-recognition Documentation*. Retrieved from Github.com: https://github.com/ageitgey/face_recognition

Karen Simonyan, A. Z. (10 Apr, 2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Retrieved from Arxiv: https://arxiv.org/abs/1409.1556

Keras. (2022). *Keras Applications*. Retrieved from https://keras.io/api/applications/

Lenne, M. (16 November, 2021). *The world is waking up to driver monitoring systems.* Retrieved from Tech Crunch: https://techcrunch.com/2021/11/15/the-world-is-waking-up-to-driver-monitoring-systems/

Makers, S. :. (24 December, 2019). *Realtime Drowsiness and Yawn Detection using Python in Raspberry Pi or any other PC.* Retrieved from https://www.youtube.com/watch?v=RDuLqCT5RxY&t=533s

Mark Sandler, A. H.-C. (21 03, 2019). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Retrieved from arxiv: https://arxiv.org/abs/1801.04381

Medel, J. R. (2016). Anomaly Detection Using Predictive Convolutional Long Short Term Memory Units . *Rochester Institute of Technology*.

Mukherjee, S. (18 August, 2022). *The Annotated ResNet-50*. Retrieved from https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

Pujara, A. (4 July, 2020). *Image Classification With MobileNet*. Retrieved from https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470

Rosebrock, A. (3 April, 2017). *Facial landmarks with dlib, OpenCV, and Python*. Retrieved from https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

Serengil, S. I. (October, 2022). *deepface Documentation*. Retrieved from Github.com: https://github.com/serengil/deepface

Shayed, A. (31 March, 2020). Retrieved from https://carro.sg/blog/5-reasons-accidents-singapore/

Shiying, W. (14 February, 2022). *Fatal road accidents in S'pore up 25% as more activities resume.* Retrieved from The Straits Times: https://www.straitstimes.com/singapore/courts-crime/more-people-died-or-got-hurt-on-spore-roads-in-2021-as-more-activities-resumed

Singapore Police Force. (2020). *Annual Traffic Statistics 2020.* Retrieved from Police.gov.sg Website: https://www.police.gov.sg/-/media/170D31BB17EF441881138E1A556F210C.ashx

Stewart, D. (14 April, 2021). *Preventing Drowsy-Driving Accidents Using Convolutional Neural Networks*. Retrieved from Towards Data Science:

https://towardsdatascience.com/drowsiness-detection-using-convolutional-neural-networks-face-recognition-and-tensorflow-56cdfc8315ad

TensorFlow. (1 October, 2022). *Model conversion overview*. Retrieved from TensorFlow: https://www.tensorflow.org/lite/models/convert/

Tsang, S.-H. (10 September, 2018). *Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015*. Retrieved from https://sh-tsang.medium.com/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

Tsang, S.-H. (25 September, 2018). *Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)*. Retrieved from https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

Wang, C.-F. (14 8, 2018). *A Basic Introduction to Separable Convolutions*. Retrieved from Towards Data Science: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728