Driver action recognition using artificial intelligence

Lee, Philip Hann Yung

2023

Project No: A3264-221

NANYANG TECHNOLOGICAL UNIVERSITY

Driver Action Recognition using Artificial Intelligence

Submitted by: Philip Lee Hann Yung

Matriculation Number: U1923382D

Supervisor: Assoc Prof Yap Kim Hui

Examiner: Prof Wen Changyun

School of Electrical & Electronic Engineering

A final year project report presented to the Nanyang Technological University
in partial fulfilment of the requirements of the degree of
Bachelor of Engineering

2023

Table of Contents

Abstract

Distracted driving is a significant cause of fatal accidents worldwide. For instance, in the United States, 9% of such accidents are attributed to distracted driving. Car-cabin monitoring solutions can help mitigate this issue by alerting drivers or authorities when instances of distracted driving are detected, potentially saving lives and increasing road safety. To address this problem, video action recognition was investigated as a possible solution for detecting distracted driving in car cabins.

A literature survey was conducted to identify relevant models and datasets, followed by hyperparameter tuning and comparison of multiple models to determine the most suitable architecture for real-time driver action recognition in terms of both accuracy and efficiency. Three state-of-the-art models were investigated, with the Temporal Shift Module-based model being recommended based on its balanced accuracy score of 68.47% on the Drive&Act dataset and a latency of 15.0 ms. Class imbalance issues were addressed through various techniques, including a fusion of class weighting and hard sample mining, resulting in a 7% improvement in baseline scores. Additionally, further work was done by fusing data from multiple modalities, resulting in a more robust model with greater prediction accuracy by further increasing scores by 5%.

The project concludes that Temporal Shift Module-based models are well-suited for real-time driver action recognition systems, which was further demonstrated by fine-tuning the model on an internally constructed dataset and successfully developing a working prototype for live testing that can run at 30 frames per second. This adds to the significance of the project as it provides a practical implementation of the proposed system.

Acknowledgements

I would like to first express my sincerest thanks to Associate Professor Yap Kim Hui, School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), for taking the role of being my supervisor. He provided me with invaluable guidance and feedback to steer the project to where it is now, motivating me to push myself further to achieve more.

I would also like to express my appreciation to Lin Dan, Research Fellow, and Li Zhengyu, Research Associate, Continental-NTU Corporate Lab. In particular, Lin Dan for being such a friendly and helpful member who was willing to go above and beyond in her work, by supporting me and providing suggestions.

Next, I also would like to thank my FYP examiner, Professor Wen Chanyun, for his time spent on assessing and examining this project.

Additionally, this project is supported under the RIE2020 Industry Alignment Fund—Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

Special thanks are extended to Martin Manuel for his assistance in answering queries related to Drive&Act benchmarking, as well as to Kunyu Peng for providing additional training guidelines for the Video Swin Transformer model on Drive&Act.

Finally, I would like to thank my family and loved ones who have supported me in thoughts, words, and actions throughout my entire undergraduate journey.

Philip Lee Hann Yung
April 2023

Acronyms

BalAcc	Balanced Accuracy Score
C3D	Convolutional 3D
CE	Cross Entropy Loss
CNN	Convolutional Neural Network
DAA	Drive&Act Dataset
FPS	Frames per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
I3D	Inflated 3D ConvNet
IR	Infrared
L2	Level 2 Fine-grained Activities
MLP	Multi-Layer Perceptron
NIR	Near Infrared
RGB	Red-Green-Blue
SGD	Stochastic Gradient Descent
SOTA	State-of-The-Art
TSM	Temporal Shift Module
TSN	Temporal Segment Network
ViT	Vision Transformer
VST	Video Swin Transformer
W&B	Weights & Biases

Symbols

c	Class index
i	Sample index
m	Modality index
K_c	Kinect RGB stream from right top location in DAA
K_d	Kinect Depth stream from right top location in DAA
K_i	Kinect Infrared stream from right top location in DAA
N_c	Number of classes
N_f	Number of frames
N_m	Number of modalities
N_s	Number of segments, or number of frames as model input
N_w	Step size of sliding temporal window
FN_c	Number of false negatives in class c
TP_c	Number of true positive in class c
$p_{i,c(,m)}$	Predicted class probability for sample i on class c (using modality m)
$s_{i,c(,m)}$	Predicted class score for sample i on class c (using modality m)
$\hat{y}_{i(,m)}$	Predicted class index for sample i (using modality m)
$y_{i,c}$	Binary indicator for sample i if actual class index $= c$
CE_i	Cross Entropy loss for sample i
f_c	Frequency of class c labels
w_c	Class weight for class c
$w_{m,c}$	Modality weight for modality m on class c
$\sigma(\cdot)$	SoftMax function

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

Distracted driving, defined as any activity that diverts attention from driving [1], may result in potentially disastrous outcomes. A 2019 study in the United States found that 9 people a day were killed and 1162 people a day were injured as a result of motor vehicle crashes by distracted drivers [2]. The same study also found that distracted driving accounted for up to 9% of total fatal accidents [2]. Undoubtedly, reducing these instances of distracted driving would consequently save a significant number of lives.

Since driving-related actions are a subset of all human actions, video action recognition can be used in a car cabin monitoring setting to detect instances of distracted driving. This would help to reduce the frequency of these distracted driving instances and consequently save lives. For example, a ringing sound could be played aloud if the system detects that the driver is using their mobile phone while driving, which would discourage mobile phone usage while driving in the long term.

As the use of autonomous driving systems becomes more widespread, current publicly deployed systems still require drivers to remain attentive while the autonomous system is in operation [3]. This is necessary so that drivers can take over when the system encounters unseen or edge cases that may pose a safety risk. As a result, there is a growing need for a real-time driver action recognition system for car-cabin monitoring.

1.2 Objectives and Scope

The purpose of the project is to explore stable and robust models for video action recognition of a driver in a car cabin environment. A frame-based approach was adopted for video action recognition, as these models can make use of visual cues of the surroundings to recognize actions. The project only considers the monitoring aspect, subsequent actions after the monitoring is out-of-scope.

The objective of this project include:

- To train and develop stable and robust deep learning-based models which can recognize driver's actions in a car cabin environment, in real-time.

The scope of this project includes:

- *Dataset*
  - Identify domain-specific open-sourced datasets suitable for driver action analysis.
  - Construct a domain-specific dataset including recording and annotations.
- *Model survey, training, and evaluation for driver action recognition*
  - Survey state-of-the-art frame-based models for video action recognition.
  - Train models on driver action related datasets and perform hyperparameter tuning.
  - Benchmark and evaluate model performance on their balanced accuracy score and perform efficiency profiling. A suitable model is recommended for a real-time car cabin monitoring system.
- *Model robustness improvement*
  - Try to enhance and improve model performance for robust recognition using a variety of techniques, including class imbalance mitigation.
  - Multi-modal fusion techniques may also be explored to increase accuracies.
- *Model deployment for a real-time car-cabin monitoring system*
  - Fine-tune the model on a constructed dataset and extend use case to develop a working prototype for live real-time demonstration.

1.3 Work Done and Contributions

The research and study of this project was done with the aim of a real-time driver action recognition system. The contributions of this project are as follows:

1. Exploration of state-of-the-art techniques

   A thorough survey was conducted to identify relevant concepts and analyze the latest frame-based video action recognition models. Additionally, suitable datasets were identified for the specific domain of car cabin monitoring solutions.

2. Extension of models to driver action recognition and benchmarking

   Three different video action recognition models were utilized to train base models on a driver action dataset. Hyperparameter tuning was performed, resulting in the training of multiple models that were compared to determine the most appropriate architecture for real-time driver action recognition, considering both accuracy and efficiency.

3. Development and improvement of model robustness

   Various training strategies and techniques were developed and compared to improve overall accuracy scores and performance. Additional work was also performed to incorporate and fuse information from different modalities, resulting in higher accuracies and implying a more robust action recognition system.

4. Dataset construction and model deployment in real-time monitoring system

   A domain-specific dataset involving 15 volunteers was constructed, with action labels and object detection boxes annotated. This dataset was used for fine-tuning a Temporal Shift Module-based model to be deployed in a real-time system. A graphical user interface was developed, and optimizations were made resulting in a successful working prototype that can run at 30 frames per second.

1.4 Organisation

The report is organized into 6 chapters. An overview of the chapters is provided below:

- **Chapter 1: Introduction** provides the project overview, including the motivations, objectives, and scope while highlighting the main contributions done.

- **Chapter 2: Background and Literature Review** reviews relevant background concepts and metrics used in the project. Recent literature and work relevant to the project were also summarized, highlighting the key ideas.

- **Chapter 3: Model Development and Considerations** discusses the modifications made to the codebase, training configurations used for the models, and proposed architectures for multi-modal fusion. The class imbalance issue was also highlighted, and solutions were proposed to mitigate it. The evaluation methodology used in this project was also detailed.

- **Chapter 4: Driver Action Recognition in Car Cabin Environments** provides an overview of the datasets used and the data preparation steps performed. The process to benchmark on an open-source dataset was discussed, and model deployment steps with modifications were made to adapt the model for use in a live real-time system.

- **Chapter 5: Results and Discussion** presents the results of various experiments conducted and recommends a suitable model for the real-time use case. Discussions were also provided such as the effectiveness of the training techniques proposed, with sample results shown for better context and an extension to a live demonstration.

- **Chapter 6: Conclusions and Future Work** provides a conclusion and summary of the project. Evaluations and recommendations for potential future work are also offered.

Chapter 2

Background and Literature Review

2.1 Background

2.1.1 Convolutional Neural Networks

Deep learning has gained much popularity in the computer vision domain, due to advancements in hardware and the exponential growth of available data, with some tasks even surpassing human-level accuracy [4]. Convolutional Neural Networks (CNNs) are commonly used for most visual recognition tasks, as they have demonstrated good results.

CNNs are a specific type of artificial neural network that contain convolutional and pooling layers, taking into account the spatial relationships between image pixels. 2D-convolution layers apply filter kernels to images to generate feature maps as an output. During the training process, these filter kernels are learned and help to extract the necessary features for downstream tasks. Figure 2-1 depicts the process of applying a convolution, in which an input image and filter generate an output tensor.

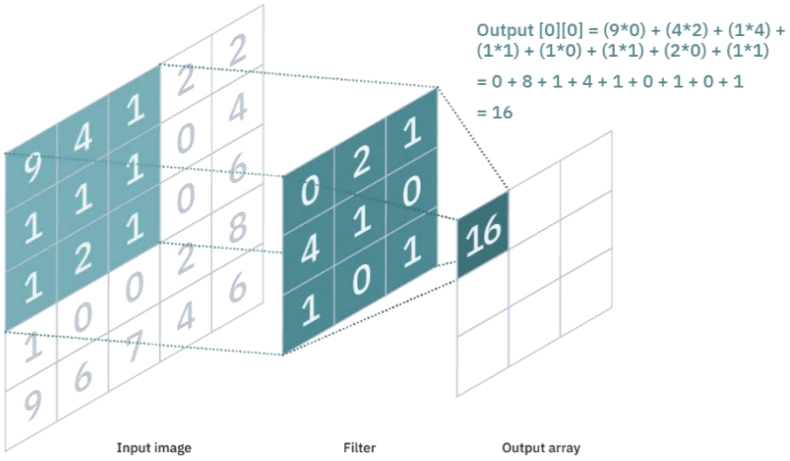

Figure 2-1: Convolution layer with a single filter kernel [5].

Additionally, pooling layers act as a downsampling layer – reducing the spatial resolution of feature maps while retaining the most important features. This reduces the number of parameters in subsequent layers. Two common pooling layers are max pooling and average pooling, in which the most important features of activation maps earlier are retained. This helps to reduce complexity thereby improving efficiency, as illustrated in Figure 2-2.

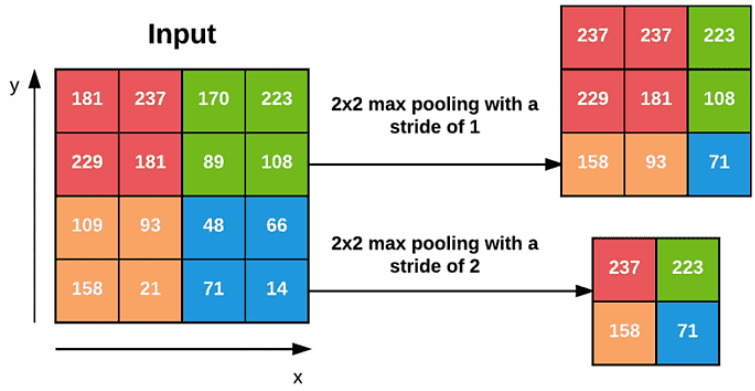

Figure 2-2: Max pooling layer with different stride sizes [6].

2.1.2 3D Convolution for Spatio-Temporal Features

While 2D convolution layers operate on multi-channel tensors to produce a single 2D feature map, 3D convolutions extend this concept using a 3D filter kernel to produce a 3D feature map as the output. This enables it to capture spatiotemporal features as concluded by [7], which is useful for video understanding since the tensors used are volumetric in nature. The authors concluded than 3×3×3 kernels perform the best and introduce a new Convolutional 3D (C3D) net that is useful for feature extraction from video data.

2.1.3 Metrics and Loss Functions

Choosing an appropriate metric for a dataset is important when reporting classification model performance, with accuracy being a commonly used metric. The model outputs predicted class scores ($s_{i,c}$) for each sample i and class c, which are then fed into a SoftMax function $\sigma(\cdot)$ to obtain class probabilities ($p_{i,c}$). A normalization factor is also used in the SoftMax function to map the probabilities to a range of [0,1].

$$p_{i,c} = \sigma(s_{i,c}) = \frac{e^{s_{i,c}}}{\sum_{c=1}^{N_c} e^{s_{i,c}}};\ \ \hat{y}_i = \underset{i \in [1, N_c]}{\operatorname{argmax}}\, p_{i,c} \tag{2.1}$$

The predicted class index ($\hat{y}_i$) is finally determined by selecting the class index with the highest probability. These series of steps are shown mathematically in Equation 2.1.

2.1.3.1 Top-k Accuracy

Accuracy may be defined as the ratio of the number of correct predictions and the total number of samples. It is relatively straightforward to calculate this for binary classification. However, for multi-class classification, it may be also interesting to find out whether the model made the correct prediction within its top k highest probabilities. Top-k accuracy assumes that classifications are correct if any of the model's top k probability classes match with the target class. Hence, the Top-k accuracy can be defined as in Equation 2.2.

For classification tasks in computer vision, popular metrics include Top-1 and Top-5 accuracy scores when comparing different models, as in the ImageNet Large Scale Visual Recognition Challenge [8].

$$\text{Top-}k = \frac{\text{Number of correct predictions within top } k \text{ highest probabilities}}{\text{Total number of samples}} \tag{2.2}$$

2.1.3.2 Balanced Accuracy

The balanced accuracy (BalAcc) or mean class accuracy metric is useful when the underlying class distribution of the data is imbalanced. The balanced accuracy may be obtained by calculating the average recall of each class. The formula is defined as in Equation 2.3, where TP_c is the number of true positives in class c, FN_c is the number of false negatives in class c, and N_c is the total number of classes.

$$\text{BalAcc} = \frac{1}{N_c} \sum_{c=1}^{N_c} \frac{TP_c}{TP_c + FN_c} \tag{2.4}$$

2.1.3.3 Cross-Entropy Loss

Cross Entropy (CE) loss is commonly used as a loss function in multi-class classification tasks. The loss value with respect to the input data is used during the update of model parameters by backpropagation – this occurs during the training process. Suppose that $y_{i,c}$ is a binary indicator for sample i if actual class index equals c and $p_{i,c}$ is predicted from the model. Then, CE loss for the sample i may be calculated using Equation 2.4.

$$\mathrm{CE}_i = -\sum_{c=1}^{N_c} y_{i,c} \log p_{i,c} \tag{2.4}$$

2.2 Video Action Recognition

Video understanding is gaining interest as it is important to understand human actions, particularly since it has many real-world applications including behaviour analysis, video analytics and surveillance. The authors in [9] provide a comprehensive review of over 200 papers on deep learning for action recognition over the past decade – a summary of the evolution of methodologies can be found in Figure 2-3.

Figure 2-3: Evolution of video action recognition methodologies.

CNNs are still widely used for video action recognition since videos are merely a sequence of images. However, the network must additionally capture some elements of the temporal component too since there is an additional temporal component/dimension for video data (represented as the number of frames), as compared to static images. In short, models should have spatio-temporal modelling capabilities. Nonetheless, video action recognition remains challenging due to the need to model activities of varying durations. For example, blinking is a simple activity that typically lasts less than a second, while walking can range from a few seconds to much longer durations.

2.2.1 Inflated 3D ConvNets

While 3D convolutions were introduced [7] at an earlier stage in 2015, the first large-scale human action video dataset Kinetics400 [10] was only released in 2017. This motivated further development in the video action recognition domain where authors of Inflated 3D ConvNets (I3D) [11] built upon existing very deep 2D CNNs trained on large image classification datasets. They were inspired by similar 2D network architectures and expanded the trained 2D convolution and pooling layers into 3D. This allowed them to leverage some features that are already learned by 2D CNNs rather than starting from a cold training, resulting in a boost in accuracy scores. The backbone used by them was an ImageNet pre-trained Inception-v1 model.

2.2.2 Temporal Segment Networks

Two-stream networks (from Figure 2-3) in their original form are unable to model long-range temporal structures, largely due to how frames are sampled from a video – as a dense stack of frames in a short duration. This dense temporal sampling strategy results in sampled frames that are highly similar since consecutive frames in a short timeframe are highly redundant. Instead, this can be replaced with a sparse sampling strategy which continues to preserve relevant information at a dramatically lower computational cost.

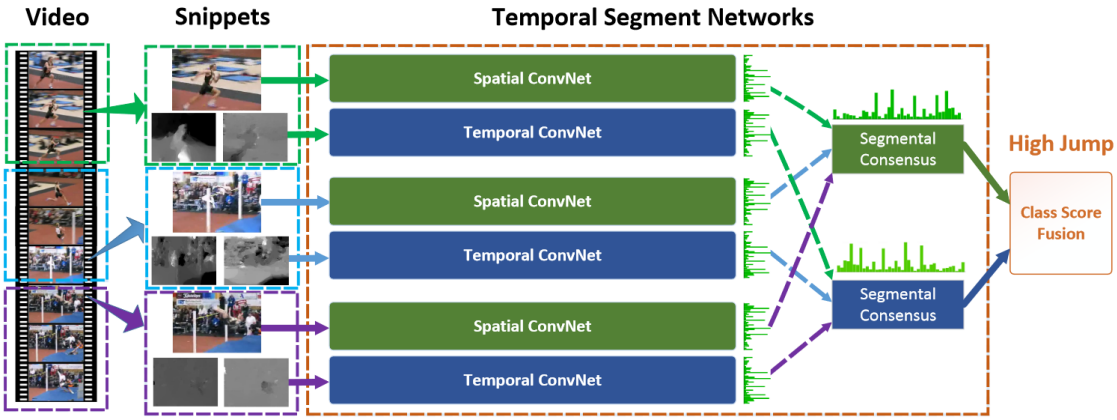

Figure 2-4: Overview of TSN framework [12].

Temporal Segment Networks (TSNs) [12] propose this sparse sampling strategy, in which a video is divided into N_s segments of equal durations. Each segment is then predicted on the same spatial and temporal CNNs to form N_s class scores. The class scores are then passed to a segmental consensus function, where average pooling was found to yield the best result, before the SoftMax function was applied to produce class probabilities. This entire process is visualized in Figure 2-4.

2.2.3 Video Swin Transformer

Transformers currently excel at Natural Language Processing tasks due to their attention mechanism which enables them to model long-range dependencies in data. To extend this idea to computer vision, the Vision Transformer (ViT) [13] model breaks down images into patches to as act input tokens. The attention mechanism allows an image patch to decide which other image patches images to attend to globally, thus having a larger receptive field than convolutional filters in the early stage.

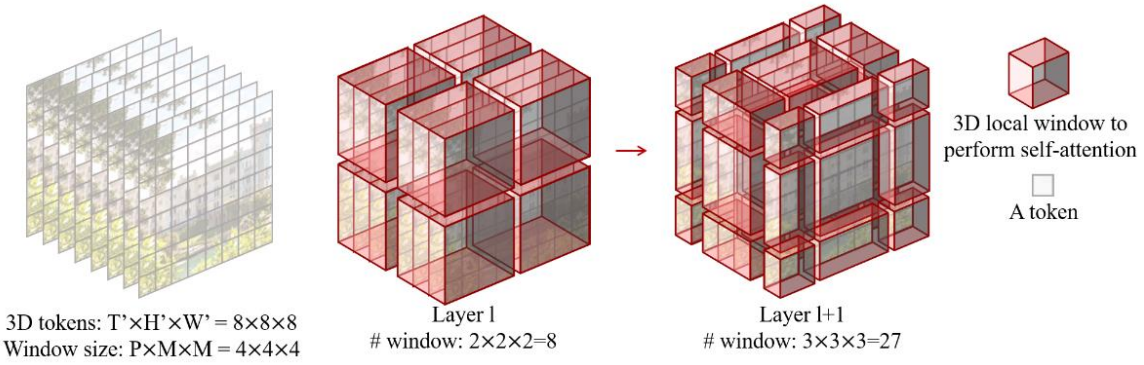

Figure 2-5: Example of 3D shifted windows in VST [14].

The Swin Transformer [15] further introduced a shifted windows scheme that reduced the self-attention operation's time complexity from quadratic (in ViT) to linear, increasing the model's efficiency. The same authors extended the idea to video data, resulting in the Video Swin Transformer (VST) [14] model, which uses 3D shifted windows. Figure 2-5 illustrates this 3D shifted windows in two layers.

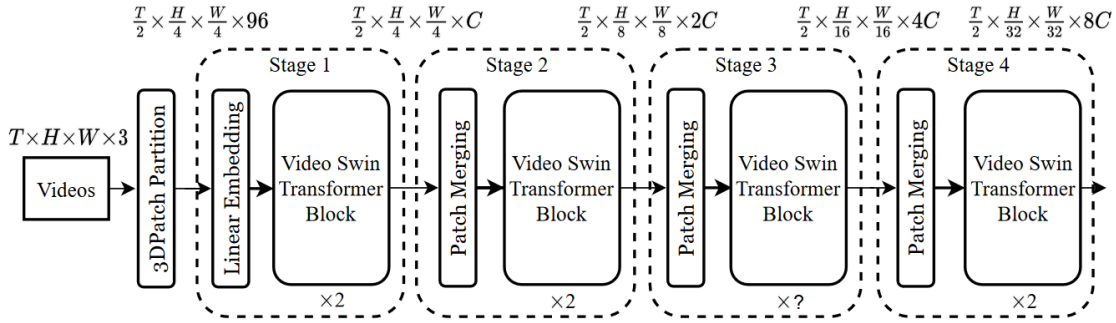

Figure 2-6: General VST architecture with 4 stages [14].

For the VST model, there are four variants officially released – tiny, small, base, and large. The general architecture is shown in Figure 2-6 with the variants differing in two ways: (i) the hidden channel number in the first stage (ii) the number of VST blocks in the third stage.

2.3 Datasets

There are many datasets available for video action recognition tasks – some samples from popular datasets like UCF101, Kinetics400 and Something-SomethingV1 are shown in Figure 2-7 . These datasets can be broadly categorized by the level of temporal modelling needed. For example, one frame in Kinetics400 is sufficient to recognize an action, like “riding a bike” by detecting the presence of the bicycle in a single frame. In contrast, movements of “dropping something” and “picking something up” in Something-SomethingV1 depend on the sequence of frames – this requires the model to have more temporal modelling capabilities.

Furthermore, since the scope is for car-cabin monitoring, appropriate datasets matching the environment used will be considered instead. The video action recognition performance in the unique car cabin environment may be influenced by different factors, such as lighting conditions, camera viewpoints, and sensor modalities used.

Figure 2-7: Examples from popular video action recognition datasets [9].

2.3.1 Near-infrared Videos

Figure 2-8: Images taken by a RGB camera (top) versus a NIR camera (bottom) [16].

Car-cabin monitoring will be active at various times of the day, so it needs to be invariant to different illuminations level. This forms the basis for the preference for near-infrared video streams over the commonly used RGB video streams in car-cabin monitoring solutions. Active NIR imaging systems produce their own NIR light source too, so it continues to function well in the dark. Figure 2-8 shows the obvious unfavourable lighting captured by a RGB camera, while it is almost non-existent in the NIR images. Hence, Microsoft Kinect devices which support infrared sensors are popular with researchers within this domain.

2.3.2 Drive&Act

The Drive&Act (DAA) dataset was released as a benchmark for fine-grained driver behaviour categorization. The dataset features over 9.6 million frames across different views and modalities (RGB, NIR, depth), making it one of the largest datasets in the car-cabin monitoring domain with high diversity involving 15 total participants.

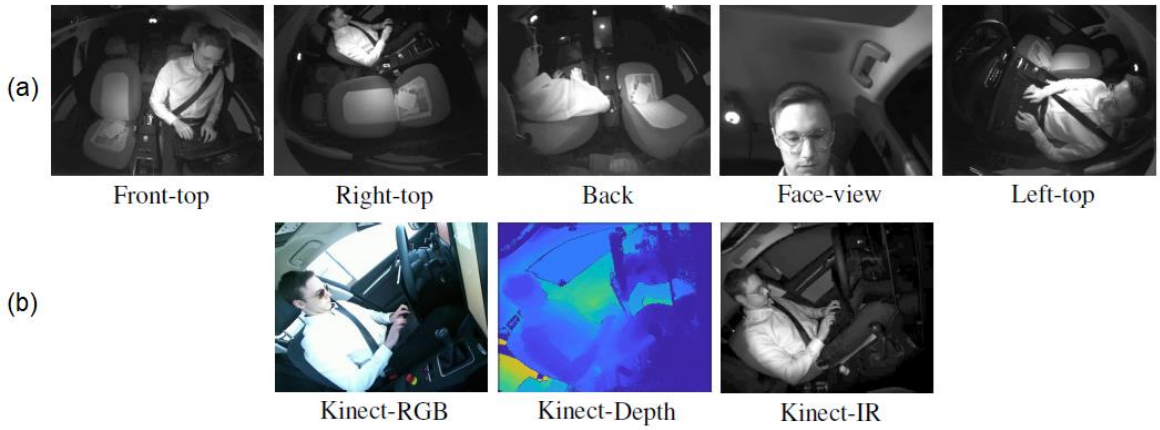

Figure 2-9: Sample images from DAA showing the "working on laptop" activity (a) NIR streams on different views (b) Different modalities on Kinect [17].

Sample images from the dataset are shown in Figure 2-9 for the "working on laptop" activity, across different views and modalities. In total, there are 83 manually annotated hierarchical activity labels provided alongside the whole dataset, with 3 train-validation-test splits provided for benchmarking purposes.

Chapter 3
Model Development and Considerations

3.1 Single Modality Video Action Recognition

This section details the training process and improvements for each model. The focus is on DAA's Front Top NIR stream which is used to determine the best performing model. The models are evaluated based on both BalAcc and efficiency to determine their performance.

3.1.1 Temporal Shift Module

The investigation begins with the exploration of a framework known as Temporal Shift Module (TSM) [18] due to its potential for high accuracy while maintaining efficiency.

3.1.1.1 TSM Framework

TSM extends 2D CNNs by taking a fraction of the channels and shifting it along the temporal dimension, resulting in neighbouring frames exchanging information between them. Figure 3-1 illustrates this shifting process, where bi-directional shift was used for offline video recognition and uni-directional shift was used for online video recognition.

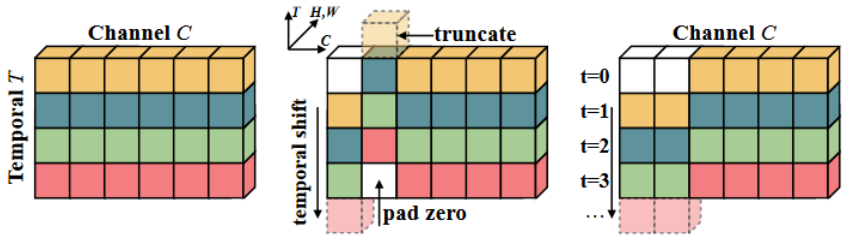

Figure 3-1: Shifting tensors along the temporal dimension

(a) Original tensor (b) Bi-directional shift (c) Uni-directional shift [18].

TSM was only applied on the residual branch of a residual block [19] with the goal of retaining the model's spatial modelling capability, instead of naively inserting it before each residual block or convolutional layer. This methodology resulted in a max Top-1 Accuracy increase of 31.3% on a ResNet50 backbone compared to TSN across various datasets, with only a marginal increase in computational cost.

By only utilizing 2D convolutional layers, pre-trained weights such as ImageNet-based ones can be used since they are widely available compared to pre-trained weights on 3D-CNN architectures. Furthermore, since 2D convolutional layers have been used in industry for quite a while, their operations have been optimized at the hardware level, enabling them to be computed more efficiently, as compared to the relatively new 3D convolutional layers.

It is important to note that the codebase of TSM was based on TSN. The sparse sampling strategy used in TSN was also used. Furthermore, many of the implementation details are found in [12] instead, so it will be necessary to refer to both TSN and TSM during the experimental phase.

3.1.1.2 Modifications on TSM Codebase

```python
class MeanClassAccuracy(object):
    """Computes and stores class predictions for mean class accuracy"""

    def __init__(self):
        self.reset()

    def reset(self):
        self.preds = []
        self.targets = []
        self.avg = 0

    def update(self, output, target):
        self.preds += output.argmax(axis=1).tolist()
        self.targets += target.tolist()

    def calc_score(self):
        self.avg = 100 * balanced_accuracy_score(self.targets, self.preds)
```

Figure 3-2: Code snippet defining the BalAcc object in TSM codebase.

The data loader script was modified to accommodate the new datasets. During initial experiments, the original TSM code saved the model with the best validation Top-1 accuracy, which was suitable for the datasets reported in the TSM paper [18]. However, during testing, the best model was saved very early in the training stage and did not necessarily result in the best validation BalAcc, Therefore, the code was modified to save the best model based on validation BalAcc instead, as shown in Figure 3-2.

In addition, Weights & Biases (W&B) [20], which is a developer-first MLOps platform with features like experiment tracking, was also integrated into the scripts for automated experiment tracking and logging. This allows for a systematic logging system of the model hyperparameter configs and results. A screengrab of the dashboard is shown in Figure 3-3, which shows the convenience that the tools provide in comparing and tracking models.

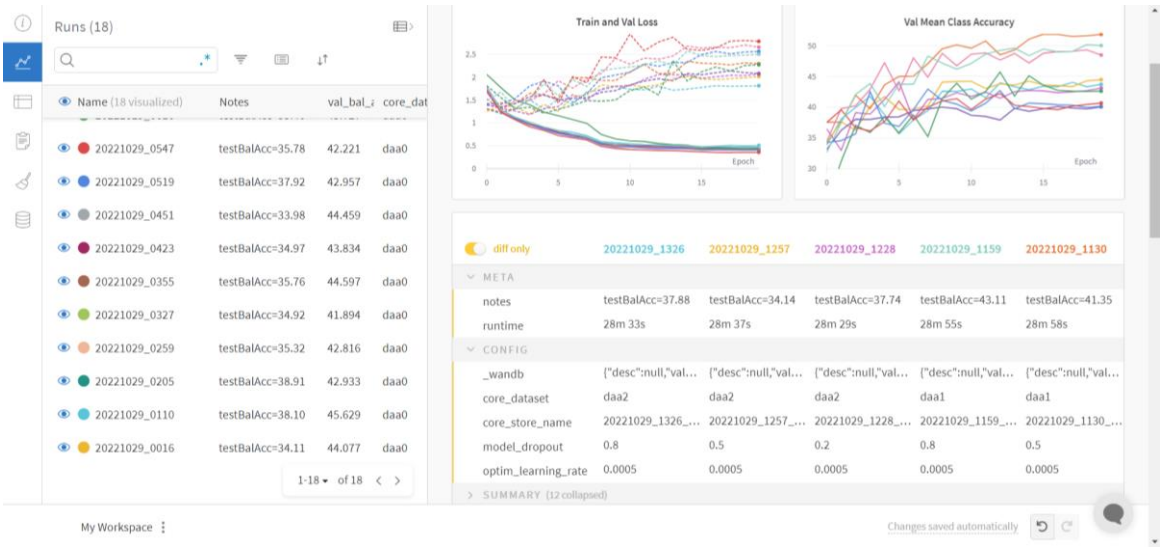

Figure 3-3: Dashboard on W&B capturing some experiments.

3.1.1.3 TSM Training Configuration and Considerations

Several parameters were explored to find the optimal configuration for the model. One observation is that the model overfitted within 10 epochs of training, requiring the learning rate (LR) to be tuned as the most important parameter. Additionally, two different numbers of segments (N_s) were tested: 8 and 16, which correspond to the number of frames used in

the model input, sampled from the total N_f. Average consensus was used to fuse scores from different segments of the video clip, as it was found to be the most effective consensus technique according to [12].

ResNet50 was selected as the feature extractor backbone for TSM, given the availability of Kinetics400 pretrained weights which may improve the generalisation capability of the model. In line with this, the corresponding Kinetics400 weights on N_s were used in comparison with ImageNet weights. Although a dropout ratio of either 0.5 or 0.8 was tested, it was found that it did not significantly affect the model's accuracy, leading to a fixed dropout ratio of 0.5 for subsequent experiments.

The training process utilized Stochastic Gradient Descent (SGD) as the model optimizer and a step decay LR scheduler with a factor of 0.1 at epochs 9 and 17, with a model input size of 224×224 px. Multi-scale cropping and random horizontal flip were used in all experiments for data augmentation. The best model was saved by monitoring the validation BalAcc over the training duration.

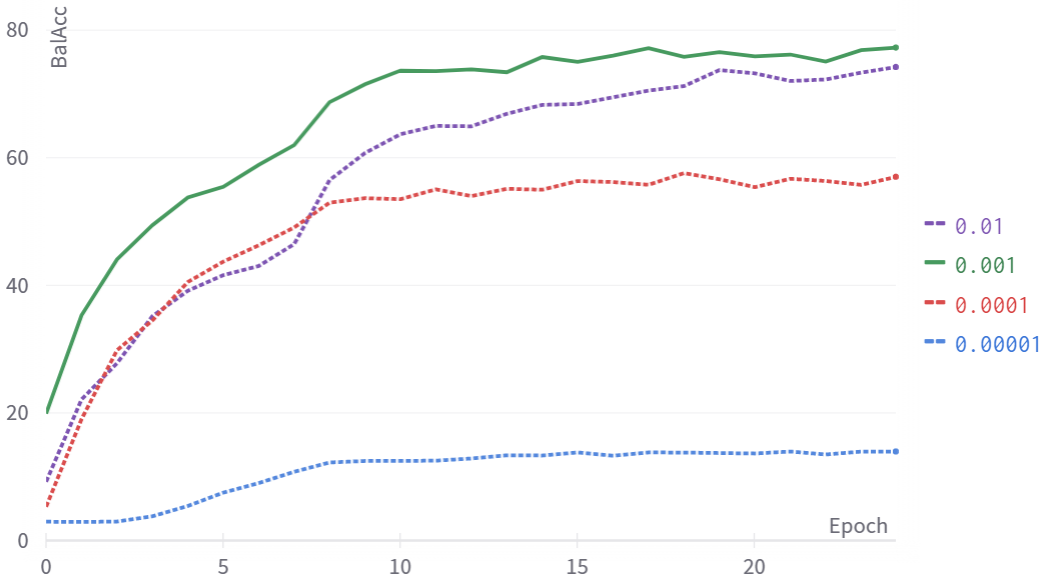

Figure 3-4: Effect of learning rate on train BalAcc of DAA split 0 using TSM ($N_s = 8$) and Kinetics400 pretrained weights, highest BalAcc is shown as solid line.

A total of 25 epochs were trained on 2 GeForce RTX 3090 Graphics Processing Unit (GPU) devices, with the batch size set to fully utilize the available GPU memory resources. Hyperparameter tuning was conducted on the learning rate to identify the optimal value. Several learning rates were set, and corresponding models were trained.

An example of such BalAcc curves on the training set of split 0 of DAA Front Top Level 2 activities is shown in Figure 3-4, using Kinetics400 pretrained weights. Only the BalAcc on the training set is displayed to avoid cluttering the figure, but the validation set BalAcc also exhibits a similar trend. It can be observed that the highest BalAcc is associated with a learning rate of 0.001. This hyperparameter tuning process was performed for all deep learning models, and the optimal learning rates for each model (TSM, I3D, VST) are summarised in this sub-section and the next two sub-sections.

Table 3-1: Optimal learning rate and the corresponding N_s and batch size for TSM.

N_s	Optimal Learning Rate	Batch Size
8	0.0010	32
16	0.0005	16

The optimal learning rates and their corresponding N_s are summarised in Table 3-1. For replication of experiments on another workstation setup, the linear scaling rule [21] may be used, in which the learning rate may be set proportional to the effective batch size (number of GPUs multiplied by batch size per GPU).

3.1.2 Baseline I3D Models

The I3D model based on the Inception-v1 backbone was presented in DAA [17] as the best performing model in 2019. However, the evaluation methodology used was different in the paper, so the performance of the model is recreated here. Thus, a similar backbone of ResNet50 was chosen to allow for a comparable and fair comparison with TSM.

The training process utilized SGD as the model optimizer and a step decay LR scheduler with a factor of 0.1 at epochs 9 and 16, for a total of 20 epochs. The batch size was set to 128 with an optimal learning rate of 0.1, with Kinetics400 pretrained weights being used.

All experiments used multi-scale cropping and random horizontal flip for data augmentation. A frame size of $N_f = 32$ and spatial dimensions of 224×224 px was used as the model input size.

3.1.3 SOTA VST Models

It may also be useful to compare the efficient TSM model with a newer SOTA model published in 2022. The performance tradeoffs will be analyzed in Section 5.1.3. Only the tiny and base variants of VST models are investigated for the purpose of this project.

The training process mostly followed the training parameters of VST on Kinetics400. AdamW [22] was utilized as the model optimizer with a cosine annealing LR scheduler using 2 epochs of linear warmup. The models were trained for a total of 20 epochs, with the batch size set to 32 and 16 for VST-Tiny and VST-Base respectively, an optimal learning rate of 0.0004, and Kinetics400 pretrained weights being used.

All experiments used random resized cropping, colour jittering and random horizontal flip for data augmentation. It was found that multi-scale cropping resulted in a slightly lower accuracy. A frame size of $N_f = 32$ and spatial dimensions of 224×224 px was used as the model input size.

3.1.4 Summary of Training Parameters

The training parameters detailed in earlier sub-sections are summarized in Table 3-2.

Table 3-2: Overview of training parameters for all models trained on DAA.

Training Parameter	TSM	I3D	VST
N_f	8, 16	32	32
Backbone	ResNet50	ResNet50	VST: Tiny, Base
Pretrained Weights	ImageNet/Kinetics400	Kinetics400	Kinetics400
Batch Size	32, 16	128	32, 16
Number of Epochs	25	20	20
Learning Rate	0.0010, 0.0005	0.1000	0.0004
LR Scheduler	Step decay: 9, 17	Step decay: 9, 16	Cosine annealing
Optimizer	SGD	SGD	AdamW

3.2 Mitigating Class Imbalance

Special consideration must be made since we are trying to maximize the BalAcc score, implying that the model must perform equally well on all classes. However, due to the class imbalance issue in the DAA dataset, the model is biased to predict more frequent classes due to their higher sample count. While this leads to a high Top-k Accuracy, the BalAcc will be lower particularly for rare classes which attains low class accuracies.

3.2.1 Class Weighting

The CE loss from Equation 2.2 may be modified to accommodate a class weight w_c such that the infrequent class samples are prioritized as more important to learning. The objective is to make all classes equally important, so w_c may be calculated such that it is inversely proportional to the frequency of class (f_c) in the training set.

$$w_c \propto \frac{1}{f_c}; \quad \mathrm{CE}_i = -\sum_{c=1}^{N_c} w_c y_{i,c} \log p_{i,c} \tag{3.1}$$

The revised CE loss is displayed in Equation 3.1, with the TSM codebase being modified to reflect this proposed change – this is shown in Figure 3-5. However, for I3D and VST, the model was trained using the MMAction2 [23] toolbox, where the functions' high level of abstraction made it difficult to perform this modification. Therefore, other methods were explored instead.

```
## EXTRA: class weighting for imbalance data, normalized such that sum is 1
class_weights = pd.read_csv(args.train_list, sep=' ').iloc[:, 2].value_counts().sort_index().to_numpy()
class_weights = np.sum(class_weights) / (len(class_weights)*class_weights)
class_weights = torch.FloatTensor(class_weights / np.sum(class_weights)).cuda()
criterion = torch.nn.CrossEntropyLoss(weight=class_weights).cuda()
```

Figure 3-5: Code snippet featuring the addition of class weights.

3.2.2 Uniform Class Sampling

Apart from the revised CE loss, modifying the data loader to randomly sample equally across classes (with replacement) can achieve the same effect as class weighting by oversampling infrequent classes. Although not all frequent class images may not be seen by the model in a single epoch (due to undersampling), they can be encountered in subsequent epochs. It is also important to note that CE loss without class weighting was used for this sub-section.

3.2.3 Hard Sample Mining

This method was tested as a standalone method and in combination with class weighting. Inspired by a technique used in [24], the rare classes may be treated as hard samples in which the model can determine automatically to train more on. To accomplish this, the training process was paused every 3 epochs. The sample-by-sample loss on the training set was computed and the mean loss was calculated.

Hard samples were defined as those samples with a loss value higher than 1.2 times the mean loss and were subsequently trained for an additional epoch to improve the model's performance on them. The regular model training was then resumed afterwards.

3.3 Multi-Modal Fusion for Robust Recognition

Multi-modality fusion involves fusing data from different modality streams with the objective of improving the model's robustness and accuracies. This may be performed at various stages of the model leading to different outcomes. The focus for this section on DAA's Right Top location using the Kinect-RGB, Kinect-IR and Kinect-Depth streams. For convenience in notation, they are represented as K_c, K_i, and K_d respectively.

The data loader, training, and evaluation scripts were modified to accommodate the multiple streams from the dataset. Much effort was involved to ensure the correct region was cropped from all 3 streams during multi-scale cropping. It is also important to note that the fusion code was inserted after loading pretrained weights. This is to make use of important features learned from other datasets.

3.3.1 Early Fusion

To fuse modality streams, one method is to concatenate them along the channel dimension, which is possible since the model input size remains constant at 224×224 px. This is illustrated in Figure 3-6 with the weights of the first convolution layer duplicated to accommodate the new input tensor size. This method allows for the pretrained weights on Kinetics400 to be used and inflated upon, retaining some level of the features learned.

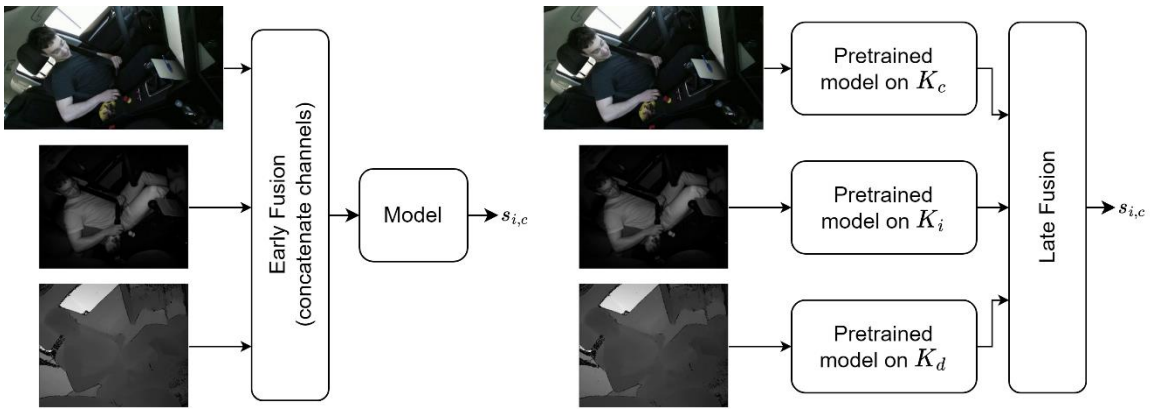

Figure 3-6: Overview of fusion architecture – Early fusion (left), Late fusion (right).

However, due to the different dimensions of the original frames between K_c and K_i/K_d, this method may not work well as the center crop regions might differ slightly and not overlap completely. Kinetics400 pretrained weights were used for training, with an identical training configuration as in Section 3.1.1.3 for TSM models.

3.3.2 Late Fusion: 3 Proposed Architectures

A late fusion technique may be more effective as the scores obtained downstream is not affected by the different dimensions. Since late fusion involves fusing the scores or probabilities predicted by each modality, modality specific pretrained models on DAA may be used – this is shown in Figure 3-6.

In all cases of this sub-section, the pretrained model is frozen, meaning that parameters will not be updated during backpropagation. Separate layers are added after the score prediction to combine the scores or probabilities. Three methods of fusing these scores are proposed and illustrated in Figure 3-7.

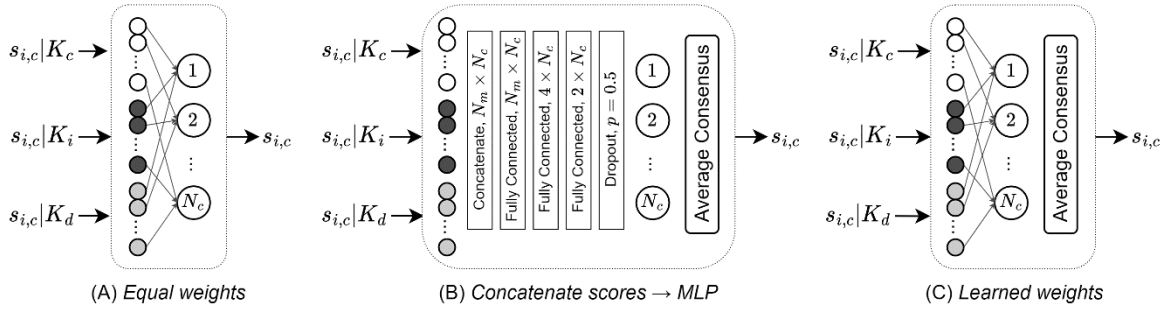

Figure 3-7: Proposed architectures for late fusion (Type A, B, C from left to right).

Since the training only involved the final few layers, the number of epochs was reduced from 25 to 20 with a step decay learning scheduler at epochs 9 and 16. Other training parameters are same as in Section 3.1.1.3 for TSM. Finally, the SoftMax function is applied to the class scores ($s_{i,c}$) to obtain the final probabilities ($p_{i,c}$).

3.3.2.1 Late Fusion: Type A

One simple way to combine the predictions from various models will be to average their predictions to form an ensemble classifier. Mathematically, this may be written as in Equation 3.2. No training was involved for this fusion method, but a combined model object was still generated and saved, combining the different pretrained models.

$$s_{i,c} = \frac{1}{N_m} \sum_{m=1}^{N_m} s_{i,c,m} \tag{3.2}$$

3.3.2.2 Late Fusion: Type B

A multi-layer perceptron (MLP) may also possibly help in fusing the scores, this type of fusion was motivated by work done in [25]. The scores ($s_{i,c,m}$) per modality are first concatenated then fed through 3 fully connected layers of size $N_m \times N_c$, $4 \times N_c$, $2 \times N_c$. A dropout layer with probability of 0.5 is used next, with the final fully connected layer having a size of N_c corresponding to the number of classes as output.

An average consensus module from TSN was then inserted to combine scores from different segments of the same clip to form the final scores ($s_{i,c}$).

3.3.2.3 Late Fusion: Type C

Averaging the predictions as in Section 3.3.2.1 assigns each modality with an equal weightage. This is not a reasonable assumption as some activities like “eating food” might not have much variation in depth during the chewing action, so the K_d modality should be assigned a lower weightage in this example.

$$s_{i,c} = \sum_{m=1}^{N_m} w_{m,c} s_{i,c,m} \tag{3.3}$$

As such, the final predicted score for each class should be instead a linearly weighted fusion of the scores produced by each modality. The model can then effectively learn the weights

($w_{m,c}$) for each modality score, on a per class basis. The formal mathematical formula for this fusion is displayed in Equation 3.3.

The implementation of this method is shown Figure 3-8, with the weight parameters initialized using Xavier initialization [26], which is commonly used as the default initialization strategy in PyTorch. To obtain the final scores ($s_{i,c}$), an average consensus module from TSN was used combine scores from different segments of the same clip. The SoftMax function was then applied to obtain final probabilities ($p_{i,c}$).

```python
class LinearWeightedAvg(nn.Module):

    def __init__(self, streams: List[str], n_neurons: int):
        super(LinearWeightedAvg, self).__init__()
        self.streams = streams
        self.n_neurons = n_neurons
        bound = 1 / math.sqrt(n_neurons)
        for stream in self.streams:
            setattr(self, f'weight_{stream}', nn.Parameter(torch.ones((1, n_neurons))))
            torch.nn.init.uniform_(getattr(self, f'weight_{stream}'), -bound, bound)

    def forward(self, x):
        output = torch.zeros_like(x[self.streams[0]])
        for stream in self.streams:
            output += x[stream] * getattr(self, f'weight_{stream}')
        return output
```

Figure 3-8: Code snippet of linear weighted scores implementation.

3.4 Evaluation Methodology

This section details the methodology used for evaluating on both validation and test sets. This ensures a fair and valid comparison across different models.

In the context of video action recognition, there are two dimensions that can be varied during model evaluation. They are usually mentioned in literature by the terminology “views”, which is in the form of (number of temporal clips × number of spatial crops). The final accuracy score is then averaged over the number of views to form a better prediction. For the efficient use case in this project, the number of temporal clips and spatial crops were both set to 1, resulting in a view of 1×1.

3.4.1 Temporal Sampling

Temporal sampling on the training set is random to serve as a form of data augmentation. However, for the validation and testing sets, frames are sampled using fixed indices. For I3D and VST, frames are sampled with a frame interval of 2 and $N_f = 32$, from the center of the video clip in the temporal dimension. If the clip length is less than N_f, the generated indices will be looped using the modulo operator by the clip length. For TSM, N_f is smaller at 8 and 16, so the sampling is performed differently. Equally spaced N_f indices are sampled from the range of 1 to the clip length. This may sometimes be a decimal, so it is then rounded down to an integer.

3.4.2 Spatial Cropping

For spatial cropping, all models follow the same procedure. The image is first resized to 256 px on the short edge, then a 224×224 px region is cropped from the center of the image. This is illustrated in Figure 3-9, where the red box highlights the region cropped.

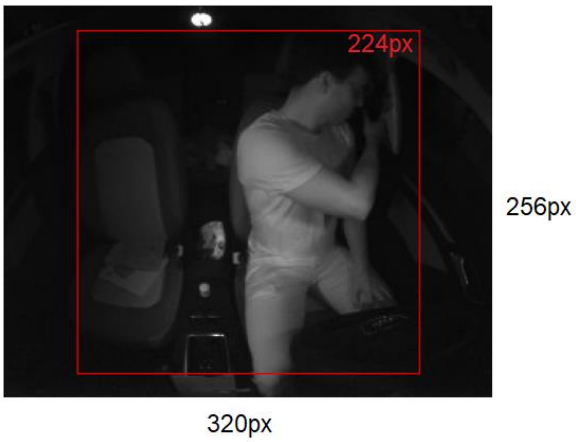

Figure 3-9: Resize and center cropping on an image.

3.4.3 Data Transformations

The data transformation pipeline, regardless of the model, is illustrated in Figure 3-10. First, frame indices are sampled, and the raw frames are read into memory. The images are then resized to a height of 256 pixels while maintaining the aspect ratio, before multi-scale

cropping (during training) or center cropping (during validation or testing) is applied to yield a fixed input size of 224×224 px.

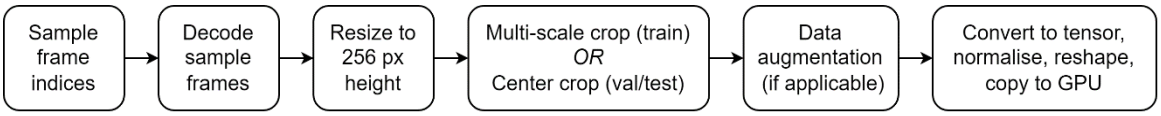

Figure 3-10: Overview of data transformation pipeline.

Data augmentations such as horizontal flipping and/or photometric augmentations are then applied. Finally, the image is converted to a tensor, normalized, reshaped, and copied to the GPU's memory.

3.4.4 Latency Profiling

Since the project application is for real-time driver action recognition in a car cabin environment, model efficiency was also of importance.

Latency experiments were performed on a GeForce RTX 3090 GPU card. The batch size was set to 1 to simulate a real-time sample-by-sample use scenario. The number of CPU workers used in data loading and data transformation was set to a large value to avoid the data loader becoming a bottleneck. This enabled an accurate measurement of the model's GPU inference time.

It is important to note that the latency profiles of the model only included the model inferencing step without data-preprocessing and other functions. Further notes on this will be detailed during real-time testing in Section 4.2.

Chapter 4
Driver Action Recognition in Car Cabin Environments

4.1 DAA Dataset

An open-source domain-specific dataset – DAA was first used for benchmarking purposes to determine the best model that may be used for a real-time driver monitoring system by evaluating various deep learning models. Specifically, Level 2 (L2) fine-grained activities were chosen for benchmarking, as these actions involve small-scale class-differentiating actions occurring within a coarser large-scale scene (e.g., vehicle cabin) which largely remains the same across frames.

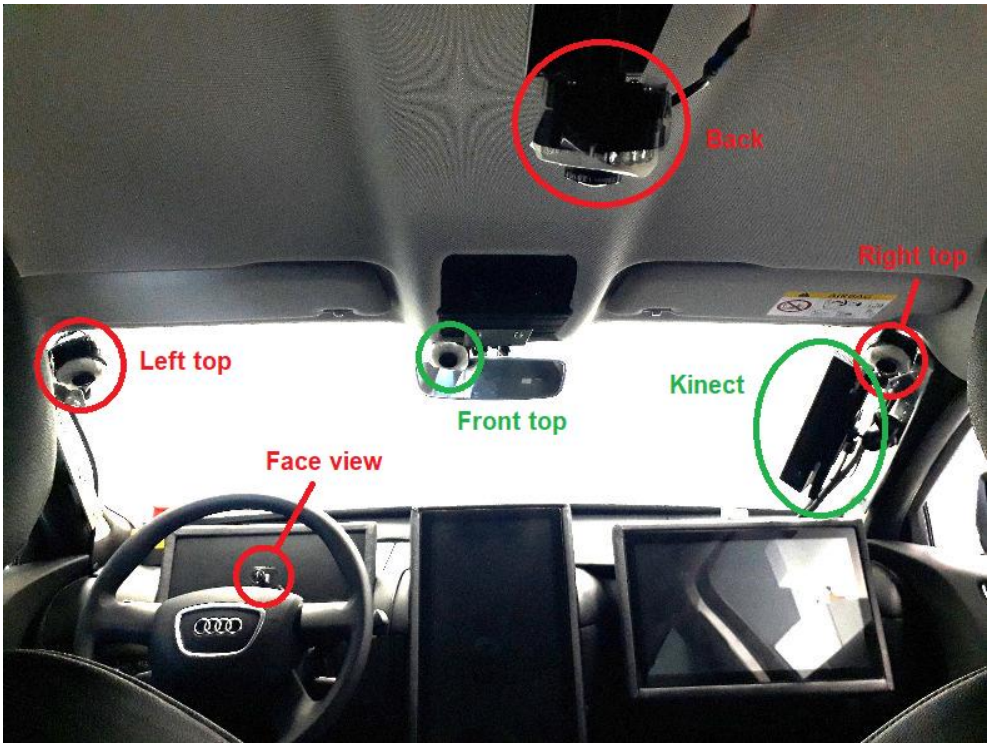

Figure 4-1: Interiors of car cabin simulator with camera positions highlighted [17].

The interior setup of the camera placements in DAA is shown in Figure 4-1, with all camera positions highlighted. Only video data from the cameras highlighted in green were used in this project, namely “Front Top” and “Kinect Right Top”.

4.1.1 Dataset Pre-processing

Since the format of expected labels was initially unknown, the label processing script from TSM was first applied on a small subset of Kinetics400 to generate a sample output file. The labels were reversed-engineered and extended to DAA in format: (samples directory, number of frames, class id). These three fields were separated by a space delimiter and saved as a text file, with each line representing one sample-label pair.

An important note is that the provided annotations file contained 39 activity classes, while the officially published number of classes for this level was 34. The discovery was made when the initial results were 10 BalAcc points lower than the reported results in the DAA paper, prompting an investigation. Upon manual inspection of the class labels listed in the paper, it was found that the 5 rarest classes were removed but not mentioned, resulting in a total number of classes (N_c) of 34 activities.

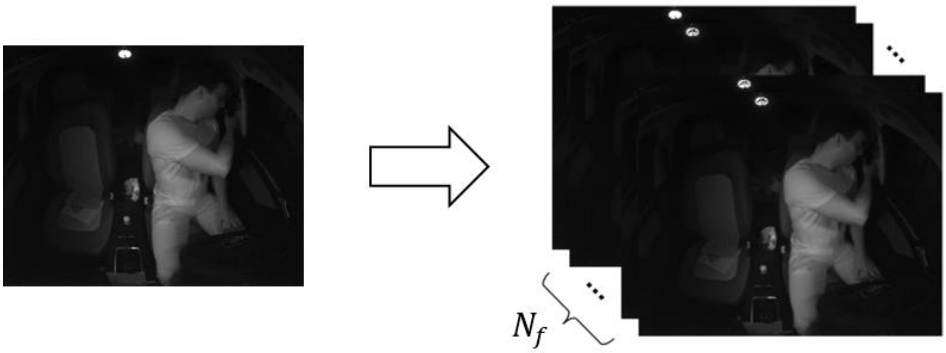

Figure 4-2: Processing video to sample clips with a fixed frame size N_f.

Since the inputs to the model must be fixed-sized, the original videos were processed into sample clips with a frame size (N_f) of 90, as shown in Figure 4-2. Activities that spanned a duration longer than 90 frames (corresponding to a duration of 3 seconds) were split into

multiple chunk samples.

Additionally, an implementation detail is that the sample clips were saved as raw frames in .jpg format. Storing the clips as frames improved the efficiency of the data loading stage during training and testing, as the video decoding process had already been performed beforehand. However, if the dataset is very large and there are resources for sufficient computational power, leaving the clips as videos may be a better option as the raw frame storage method requires more storage space.

In terms of spatial resolution, the frames were resized to a height of 256 px to reduce storage requirements. This operation could be performed at this initial stage since it is an identical data transformation step during model training and evaluation. As a reference, the 29 original videos in Front Top occupied 2.1 GB, while the corresponding 778,059 labelled raw frames occupied 7.4 GB.

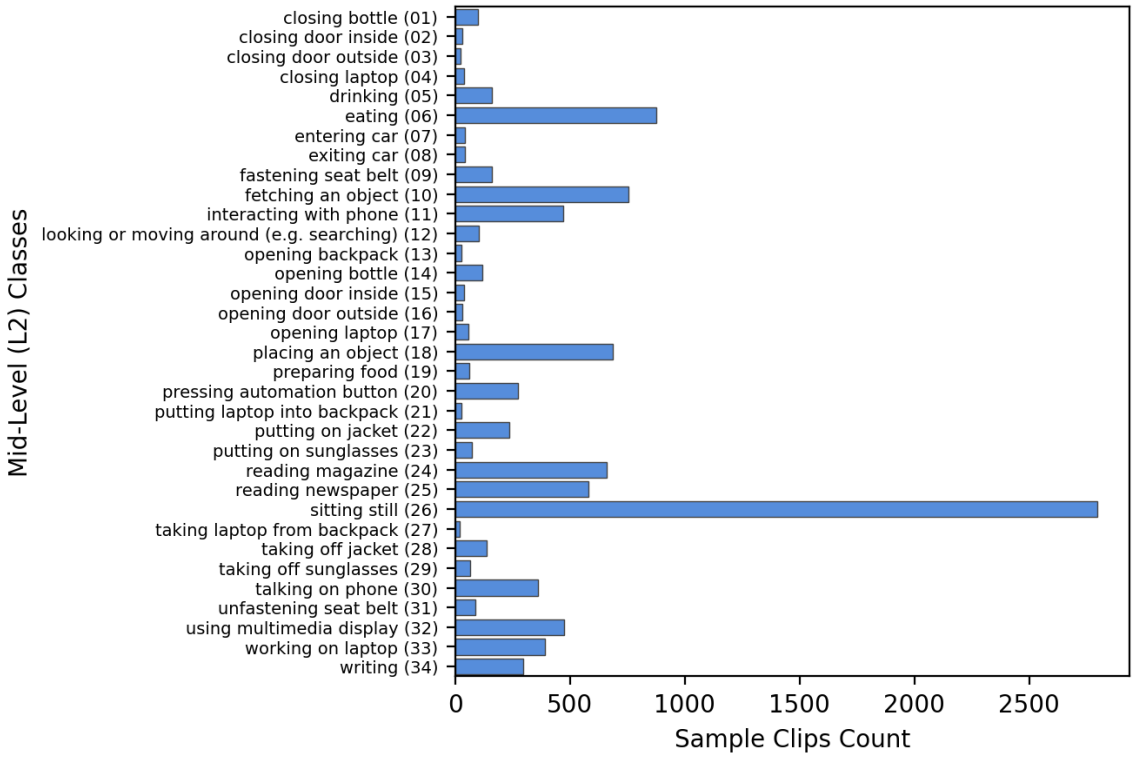

Figure 4-3: Distribution of the number of samples for each class after processing using L2 activity labels from DAA Front Top location.

After processing the videos to sample clips, a total of 10294 video samples were obtained from the Front Top location. Figure 4-3 shows the count of the number of samples for each class, with the lowest count of 19 samples for "taking laptop from backpack" and the highest count of 2797 samples for "sitting still". The figure also reveals a class imbalance on the dataset. Hence, the BalAcc metric would be more suitable compared to Top-k Accuracy, which is supported as the BalAcc metric was reported by DAA authors when comparing models.

4.1.2 DAA Splits

The DAA dataset was randomly divided based on the driver identity [17], which enables the model to be evaluated on unseen people. A total of 3 unique splits of train-validation-test (train-val-test) were provided as a form of 3-fold cross validation on the model. This helps to better ensure that the model's performance is not dependent on a particular train-val-test subset.

```
# Master split dictionary
splits_dict = {
    0: {'train': [1,2,3,4,6,7,8,9,10,12], 'val': [14,15], 'test': [5,11,13]},
    1: {'train': [1,2,4,5,6,7,11,13,14,15], 'val': [3,8], 'test': [9,10,12]},
    2: {'train': [3,5,8,9,10,11,12,13,14,15], 'val': [1,2], 'test': [4,6,7]},
}
```

Figure 4-4: Code snippet form the pre-processing script showing the different DAA splits.

3 sets of annotations were subsequently produced for each location, using the details mentioned in Section 4.1.1. The specific driver id for train-val-test in the 3 splits is shown in Figure 4-4. It is important to note that the train-val-test subsets are mutually exclusive to prevent data leakage issues.

Only the Front Top and Right Top locations were used in this project. The Front Top location was used for training and evaluating single-modality models, while the Right Top location was used for multi-modality models. Table 4-1 summarizes the number of samples for each split and these locations.

Table 4-1: Breakdown of sample counts for each split on DAA L2 activities.

Location	Split	Train	Val	Test	Total
Front Top	0	6642	1430	2222	10294
	1	7253	1385	1656	10294
	2	6693	1356	2245	10294
Right Top	0	6746	1459	2253	10458
	1	7374	1404	1680	10458
	2	6796	1375	2287	10458

4.1.3 DAA Benchmarking

The methodology for benchmarking the DAA dataset was not initially clear and was only known after further discussions with the DAA authors. In summary, the benchmarking process involves training three separate models on each split (0, 1, 2). The predicted labels for each split are obtained by inferencing the validation or test set on their respective model.

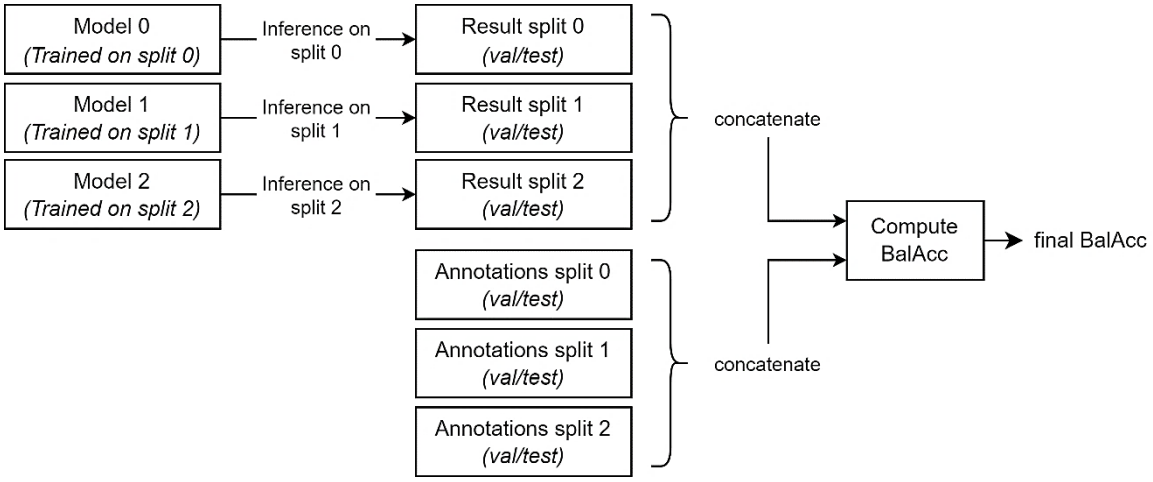

Figure 4-5: Methodology for calculating the combined BalAcc score on DAA.

These results and annotations are then concatenated across the three splits to produce a consolidated version, which is used to compute the final BalAcc score. Figure 4-5 provides a clear illustration of this process.

4.1.4 Resolving Multi-modality Issues

The Right Top camera location was used since 3 modality streams are provided from the Kinect sensor. As a reference again, Kinect-RGB, Kinect-IR, and Kinect-Depth are represented as K_c, K_i, and K_d respectively.

Before multi-modal fusion could be performed, two main issues needed to be resolved. The first issue was synchronizing frames between the different modalities using the provided annotation timestamps. The K_i and K_d labels were found to be identical, requiring no further action. However, for K_c and K_i, it was observed that K_c labels were sometimes delayed by a few frames compared to K_i. Therefore, an offset was calculated for K_c from the K_i labels to ensure consistency, with K_i being used as the ground truth annotations.

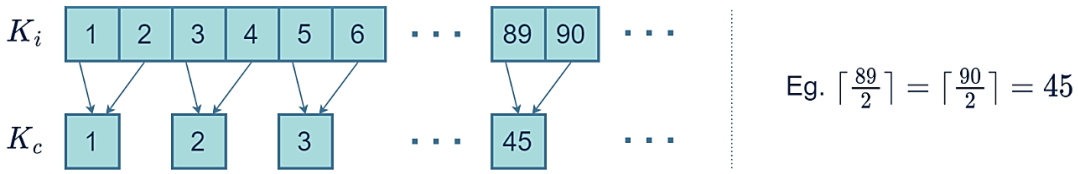

Figure 4-6: Generating new indices for K_c.

The second issue stemmed from the different sampling rates of the sensors used in the Microsoft Kinect for Xbox One. K_i and K_d was recorded at 30 Hz (512×424 px), while K_c was recorded at 15 Hz (950×540 px). This implies that for every 2 K_i/K_d frames, the same singular K_c frame may be mapped to it. A naive splitting of K_c into video samples with $N_f = 45$ (half the initial N_f) was attempted, but it resulted in a total number of samples that are inconsistent with K_i and K_d.

$$\lceil \text{Total activity length in } K_i \,/\, 2 \rceil = \text{Total activity length in } K_c \qquad (4.1)$$

To address this issue, the official K_c annotations provided were discarded, and new annotations were generated by mapping every 2 labels from K_i to 1 label on K_c, considering the calculated offset earlier. The goal was to ensure that the total activity length

in K_c was consistent with the total activity length in K_i, with the formula in Equation 4.1. The process of generating these annotations is illustrated in detail in Figure 4-6, which includes frame indices for clarity.

After all the issues had been resolved, the three modality videos are pre-processed as outlined in Section 4.1.1. They were split into sample clips and stored as raw frames.

4.2 Live Testing and Demonstration

To adapt the project's use case to a working real-time monitoring system prototype, the model was fine-tuned on an internal dataset recorded in Continental-NTU Corporate Lab.

4.2.1 Dataset Construction and Training

An experimental dataset was created by recording the RGB stream of a Kinect sensor in a simulated car cabin environment located in the lab. The decision to use RGB instead of NIR was due to dataset requirements for other projects. A total of 15 volunteers (including FYP students) performed various activities and interacted with objects similar to those in Level 2 DAA activities, in both daytime and night-time scenarios. The dataset was labelled with 32 action classes, and each volunteer annotated object detection bounding boxes themselves. Due to confidentiality requirements, the complete list of activity labels cannot be disclosed.

The simulated car cabin environment setup is shown in Figure 4-7, with the position of the Kinect device highlighted in red. The curtains in the background of the photo can also be drawn back to simulate changes in lighting conditions to represent daytime or night-time scenes.

The dataset was pre-processed similarly as detailed in Section 4.1.1. However, due to the smaller size of the internal dataset compared to than DAA, the samples were only divided into training and validation subsets. 3 out of the 15 volunteers were placed in the validation set, including Philip (the demo person) to prevent data leakage during real-time testing.

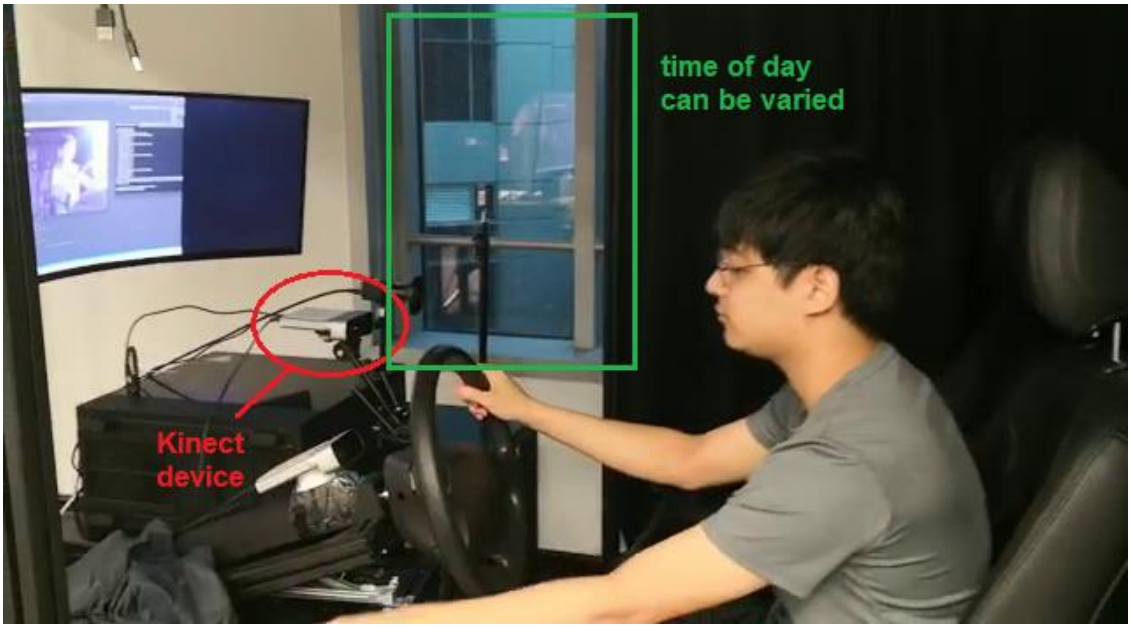

Figure 4-7: Side view of car cabin environment setup in lab.

4.2.2 Additional Modifications

The demo script performance was enhanced through two main modifications to increase its robustness and reduce latency.

4.2.2.1 Right Square Cropping

The image obtained from the Kinect RGB stream is first resized to a height of 256 px, similar to the methodology. The width-height ratio can be defined as the ratio of the width to the height. A higher ratio indicates that the image is wider horizontally. For the Front Top location in DAA, the ratio is 1.25 (from 1280×1024 px), while the demo image has a ratio of 1.78 (from 1280×720 px).

The larger width-height ratio causes issues if the center cropping evaluation methodology is used. This is illustrated in the left of Figure 4-8, where the cropped region in red excludes a significant portion of the driver. This issue is not present in videos from the Right Top location of DAA because the central region of the frame contains most of the driver.

Figure 4-8: Visualization of cropped region in red – Center cropping (left),
Right square cropping then center cropping (right).

Consequently, a square cropping step on the right side was added, which is illustrated by the region to the right of the green line in the second figure of Figure 4-8. This results in a center cropped region that contains much of the driver. This “Right Square Cropping” step was added as the first transformation step to the dataloader functions of both train and evaluation script. During training, the multi-scale cropping occurs from within the right square region of the image. Since the project scope is limited to recognizing the actions of the driver only, other passengers or locations in the car cabin are not considered relevant.

4.2.2.2 Optimized Data Transformations

While the latency of the TSM model was low, the data transformation or pre-processing step was identified as the main bottleneck after subsequent testing, taking 124 ms. The main contributing factor to this, which took 94.1 ms, was the conversion of a NumPy array, used by OpenCV, to an internal data type utilized by Python Imaging Library. Since the data obtained from the camera was in the form of a NumPy array, converting all the transformation functions to be NumPy-based is a more appropriate and efficient.

Further optimizations could be achieved by removing logical checks and extracting the main transformation functions, since the format and data type of the input data are known. For instance, the function only needs to process only one video sample at a time due to the real-time nature of the system, so a generalized function is not needed.

```python
def transform_fast(frames_list):
    data = [frames_list[i][:, -H:, ::-1] for i in indices]  # Right cropping
    data = [cv2.resize(img, (256, 256))[16:-16, 16:-16, :] for img in data]
    data = np.concatenate(data, axis=2)
    data = torch.from_numpy(data).permute(2, 0, 1).contiguous()
    data = data.cuda(0).float().div(255)
    data = normalize_transform(data)
    return data.unsqueeze(0)
```

Figure 4-9: Code snippet featuring optimized data transformation function.

The optimized version of the data transformation function is shown in Figure 4-9, which offers an almost 8-fold improvement in data transformation times, requiring only 15.6 ms.

4.2.3 Demo Environment

The real-time system was deployed on a workstation that was equipped with a single GeForce RTX 3080 GPU card, different from the server used to train the models. A demo script was developed to perform inference within a sliding temporal window of size $N_f = 90$ that has a step size N_w. This means that frames are saved in a buffer queue of size $N_f = 90$, but only predicted every N_w frames – this is illustrated in Figure 4-10.

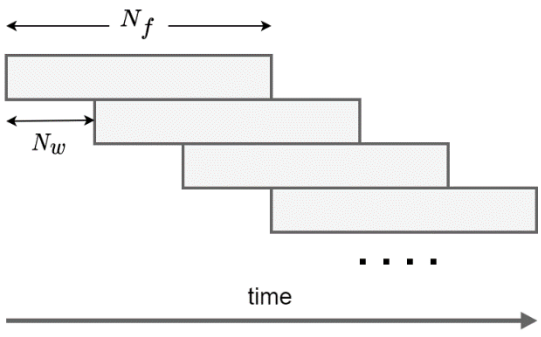

Figure 4-10: Sliding temporal window of size N_f with step size N_w.

The same Kinect camera was positioned at the same location as in the dataset, and the demo made use of the same RGB stream.

Chapter 5
Results and Discussion

5.1 Single Modality Results

5.1.1 Base TSM Results

6 TSM models were trained corresponding to 3 DAA splits each using ImageNet or Kinetics400 pretrained weights. The training and validation loss and BalAcc curves in Figure 5-1 reflects the experimental process of a particular run. The figure clearly shows a significant jump between epoch 8 and 9, after the curves appear to plateau just before it. This emphasizes the significance of the step decay learning rate scheduler which reduces the step size during the backpropagation stage when it is already in the proximity of the local minima during optimization.

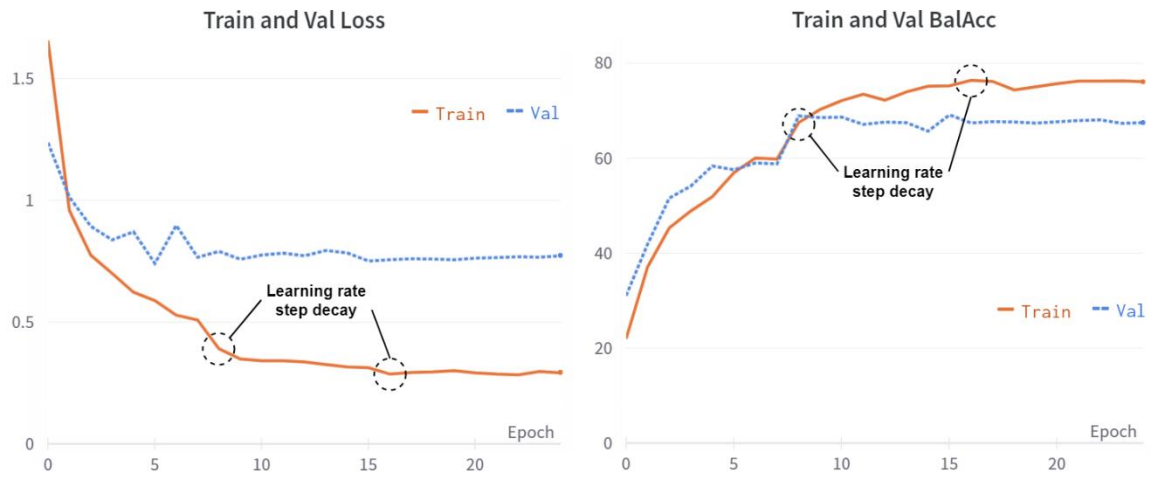

Figure 5-1: Training and validation loss and BalAcc curves on a sample run, the epochs with learning rate step decay are highlighted showing its effectiveness.

Table 5-1 presents the results while varying the pretrained weights. Changing the pretrained weights from ImageNet to Kinetics400 increased the validation BalAcc by 8-9 points. This

implies that some temporal features were learned from the Kinetics400 dataset as the Kinetics400 dataset poses as a video action recognition problem rather than a static image classification for the ImageNet dataset, which also validates the findings in [11]. As a result, subsequent model training was performed using the Kinetics400 pretrained weights.

Table 5-1: Comparison of results on DAA using different pretrained weights on TSM.

Model	Pretrained Weights	N_s	Val ↑	Test ↑
TSM	ImageNet	8	52.17	42.98
		16	51.77	43.34
TSM	Kinetics400	8	60.17	55.24
		16	**60.64**	**56.76**

The TSM model trained using Kinetics400 pretrained weights is referred to as “baseline”.

5.1.2 Improved TSM Results

All three proposed methods for addressing class imbalance issues were tested on TSM. The results are summarized in Table 5-2 for both N_s values of 8 and 16, with class weighting surfacing as the most effective standalone method for TSM.

Table 5-2: Comparison of results on DAA while varying training methods on TSM.

Model	Training Method	N_s	Val ↑	Test ↑
TSM	Baseline	8	60.17	55.24
		16	60.64	56.76
TSM	Class Weighting	8	67.31	61.90
		16	65.98	59.41
TSM	Uniform Class Sampling	8	64.33	58.13
		16	64.57	57.79
TSM	Hard Sample Mining	8	61.09	56.67
		16	62.38	55.43
TSM	CW+HSM	8	**68.47**	**62.34**
		16	66.57	59.37

The hard sample mining approach potentially treats the rare classes as hard samples, but it is not enough to address the problem of infrequent classes. While this method improves the balance accuracy compared to the baseline approach, it is still less effective than using class weighting or uniform class sampling. However, when combined with the best standalone class imbalance solution, i.e., class weighting and hard sample mining concurrently (CW+HSM), this approach produced the best validation and test BalAcc scores, achieving 68.47 and 62.34, respectively, with a further one-percentage point boost.

Figure 5-2: Sample predictions from TSM with CW+HSM on DAA test set split 0 with ground truth labels “talking on phone” (top), “eating” (bottom).

The prediction of actions on clips from DAA test set split 0 was performed using TSM with CW+HSM. The results were displayed with the top 5 predictions sorted in descending order of confidence scores and located on the left top corner of the video clip. Examples of the

clips are shown in Figure 5-2. In the top row of images, the model accurately predicts the ground truth label of “talking on phone”, which is useful since it can detect instance of distracted driving. However, the bottom row of images has a ground truth label of “eating”, but the model’s accuracy is dependent on the presence of food in the sampled clip. If the food object is absent but chewing is still present, the model may not detect it if the chewing motion is not prominent, which may be due to how the eating activity was defined and labelled in DAA.

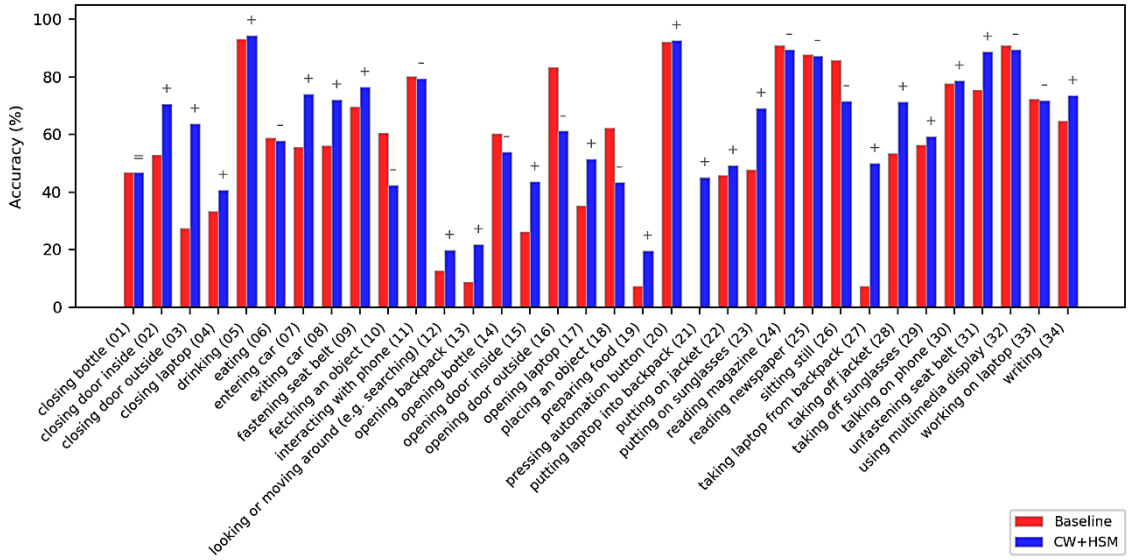

Figure 5-3: Comparison of class accuracies between baseline and CW+HSM on DAA combined test set (relative change from baseline: = no change, + increase, – decrease).

To emphasize the importance of mitigating class imbalance, the difference in the class accuracies is shown in Figure 5-3 between the baseline and CW+HSM model on the combined test set. For this example, it can be observed that 22 classes (out of 34 total classes) have demonstrated an increased class accuracy, indicating an improved classifier that does not favor predicting overrepresented classes.

Addressing the class imbalance issue ultimately leads to a different objective function, which could explain why the model using $N_s = 16$ does not consistently perform better than $N_s = 8$. The model using a higher N_s may be overfitting to the training set due to its

larger theoretical ability in modelling temporal features. This may be due to the DAA dataset not being as diverse as larger datasets such as Kinetics400.

As a final note, since TSM with CW+HSM and $N_s = 8$ yields the best results, it is used for comparison with other models, multi-modal fusion, and training the model for the real-time demo in subsequent sections.

5.1.3 Comparison between Models

The results of I3D, TSM, VST trained on 3 splits are summarized in Table 5-3. As expected, the VST-Base model achieves the best performance due to its deeper architecture and better spatiotemporal modelling capabilities. Both baseline and class imbalance mitigated training methods are presented, which once again highlights the importance of matching the BalAcc metric reported with an appropriate training method. This can be observed where the baseline method yields a consistently lower score.

Table 5-3: Summary of BalAcc scores on DAA between trained models.

Model	**Training Method**	N_s	**Val ↑**	**Test ↑**
$I3D_{ResNet50}$	Baseline	32	56.22	53.79
	Uniform Class Sampling		63.29	60.41
$TSM_{ResNet50}$	Baseline	8	60.17	55.24
	CW+HSM		68.47	62.34
VST_{Tiny}	Baseline	32	64.05	59.02
	Uniform Class Sampling		68.67	60.10
VST_{Base}	Baseline	32	66.44	61.89
	Uniform Class Sampling		**70.06**	**63.48**

While accuracy results are important, it is more interesting to compare the BalAcc scores along with the models' efficiency profiles shown Table 5-4. Since TSM is a newer method compared to I3D, it indeed performs better with a similar latency magnitude. The latency of TSM is also the lowest which makes it particularly suitable for a real-time use case.

The throughput of the models is also reported by calculating the inverse of the latency times, which results in units of clips per second. This measurement indicates how many sample clips of size N_f the model can process in one second. The conventionally used frames per second (FPS) is not applicable for this purpose, as the input data is in the form of video. However, FPS is still reported during the real-time demo, as it directly relates to the number of frames processed by the real-time system.

Table 5-4: Efficiency profiles across different models.

Model	Latency ↓	Throughput ↑	GPU Memory ↓
$I3D_{ResNet50}$	18.3 ms	54.7 clips/s	2.51 GB
$TSM_{ResNet50}$	**15.0 ms**	**66.7 clips/s**	**2.00 GB**
VST_{Tiny}	40.2 ms	24.9 clips/s	3.40 GB
VST_{Base}	89.3 ms	11.2 clips/s	4.16 GB

While the VST-Base model performs the best, the latency is also the highest at 89.3 ms which is 6 times slower than TSM, thus it's application will be more suitable on systems without real-time response requirements. On the other hand, VST-Tiny model may be used as an intermediate alternative with moderate latency and a good accuracy.

Since TSM has the lowest latency with a comparably high BalAcc, it is recommended to be used for a real-time system like driver action recognition in a car cabin environment.

5.2 Multi-modal Fusion Results

By fusing features to exploit information from different modalities, the models' robustness can be improved, enabling them to accurately classify various activities under diverse environmental conditions, resulting in more reliable and valuable results.

The experiments conducted on TSM using the Front Top location were repeated with the Right Top location of DAA, now with 3 different modalities – K_c, K_i, K_d. The fusion methods proposed in Section 3.3 were also applied using all combinations of the 3

modalities, with the training method incorporating CW+HSM too. These results are summarized in Table 5-5, with the first row presenting results without any fusion.

Table 5-5: Summary of proposed multi-modal fusion methods on DAA while varying modality combinations.

Fusion Method	**Modality**	**Val ↑**	**Test ↑**
No Fusion (single modality)	K_c	70.35	62.72
	K_i	69.33	59.81
	K_d	68.31	58.28
Early Fusion (concatenate channels)	$K_c + K_i$	69.12	63.46
	$K_c + K_d$	66.08	59.97
	$K_i + K_d$	69.74	60.44
	$K_c + K_i + K_d$	66.50	61.30
Late Fusion: Type A (average probabilities)	$K_c + K_i$	73.52	65.30
	$K_c + K_d$	73.25	65.19
	$K_i + K_d$	73.24	62.87
	$K_c + K_i + K_d$	73.81	64.69
Late Fusion: Type B (concatenate scores - MLP)	$K_c + K_i$	70.32	63.06
	$K_c + K_d$	66.50	61.30
	$K_i + K_d$	72.50	62.51
	$K_c + K_i + K_d$	73.42	64.60
Late Fusion: Type C (linear weighted scores)	$K_c + K_i$	71.47	64.99
	$K_c + K_d$	74.06	65.09
	$K_i + K_d$	74.02	63.24
	$K_c + K_i + K_d$	**75.20**	**66.32**

In the case of Early Fusion, all results obtained using the validation set were lower than the highest single modality K_c BalAcc score of 70.35. It is hypothesized that concatenating features in the channel dimension in a naive manner may not be such a straightforward approach, and that a more advanced technique needs to be used. In particular, when the K_c modality was added to the other modalities, the scores decreased. This may be due to the K_c image frame not being from the same viewpoint and dimensions as K_i/K_c, which may adversely affect the spatial feature learning of the model.

In the case of Late Fusion Type A, the ensemble model performs as expected, with the average scores from more modalities generally leading to a better score. This is reflected where $K_c + K_i + K_d$ achieved the best validation BalAcc at 73.81, representing a 3.5-point increase in the BalAcc score.

In the case of Late Fusion Type B, the model did not perform as well as the other proposed late fusion methods. This may be due the concatenation of scores, which resulted in the loss of direct class information since the MLP layers have a receptive field that included all the scores. Consequently, the Late Fusion Type C method was proposed to preserve the scores conditioned on the class, with the class score $s_{i,c}$ being a function of the scores of the same class across different modalities ($s_{i,c,m}$).

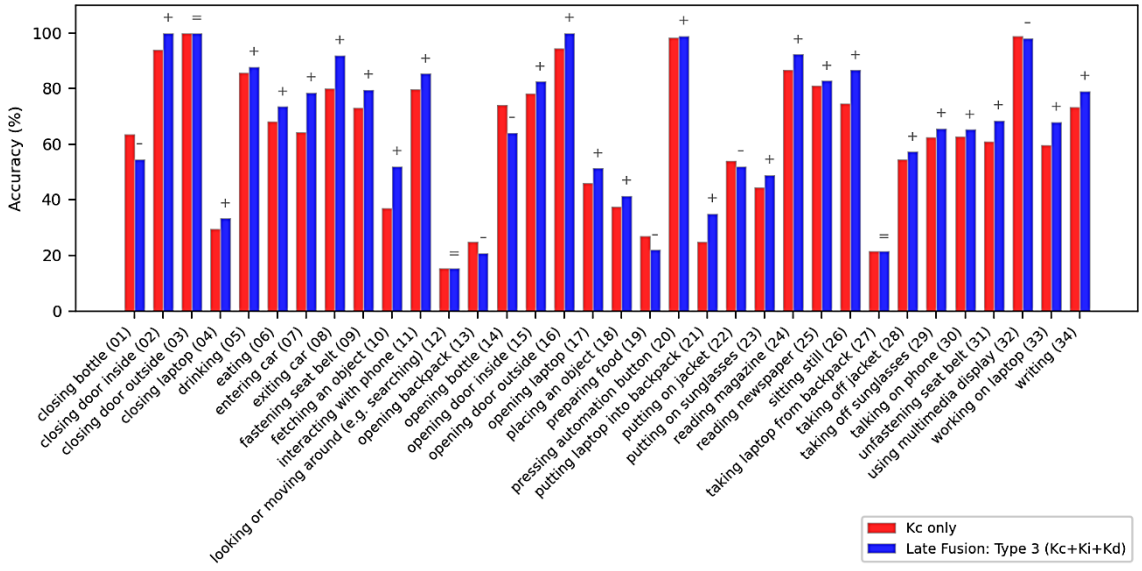

Figure 5-4: Comparison of class accuracies between the best single modality model (K_c) and best multi-modality model (Late Fusion: Type 3 using $K_c + K_i + K_d$) on DAA combined test set (relative change from K_c only: = no change, + increase, – decrease).

Finally, in the case of Late Fusion Type C, the model generally performed better than the other proposed fusion methods, with one exception: $K_c + K_i$, where the scores were lower than Late Fusion Type A. This may be attributed to the K_c and K_i streams being very

similar and not providing sufficient different perspectives to result in a significant improvement in BalAcc. This is in contrast to the addition of K_d to K_c/K_i, where the depth information provided a distinct perspective from the same scene.

From the results in Table 5-5, it can be observed that the best validation and testing BalAcc scores of 75.20 and 66.32, respectively were achieved by a weighted late fusion (Late Fusion Type C) of $K_c + K_i + K_d$ scores. The difference in class accuracies provided by this fusion approach compared with K_c is shown in Figure 5-4, with 25 out of 34 classes showing an increase in accuracies. This highlights the potential of the proposed fusion model to provide more reliable predictions.

Table 5-6: Efficiency profiles using multiple modalities in Late Fusion Type C.

N_m	Latency ↓	Throughput ↑	GPU Memory ↓
1	**15.0 ms**	**66.7 clips/s**	**2.00 GiB**
2	33.0 ms	30.3 clips/s	2.16 GiB
3	46.1 ms	21.7 clips/s	2.36 GiB

Latency profiling was also performed for Late Fusion Type C models as shown in Table 5-6, with $N_m = 2, 3$ having a latency of 33.0 ms and 46.1 ms respectively. The $K_c + K_d$ and $K_c + K_i + K_d$ models are therefore recommended since they still adhere to the real-time requirement and enjoy a significant boost in accuracy.

5.3 Real-time Monitoring System

As discussed in Section 5.1.3, the TSM model was identified as the most appropriate model for real-time driver action recognition. The TSM model was trained with a batch size of 32 for 25 epochs, using the best learning rate of 0.0025 and other training parameters consistent with those outlined in Section 3.1.1.3. Both class weighting and hard sample mining methods were also used during training.

The TSM model achieved the highest BalAcc score of 60.19 on the validation subset of the internal dataset. After training, the model was used and loaded to demonstrate a real-time monitoring system. The optimal temporal window step size N_w was found to be 30 frames, resulting in predictions that update every 1 second for a 30 FPS video camera. This value of N_w was chosen to provide frequent prediction updates while retaining the ability to recognize actions that span a longer duration.

As explained in Section 4.2.2.2, the optimized data transformation for each frame took 15.6 ms instead of 124 ms. Table 5-7 summarizes the latency times for each stage of the prediction pipeline for one sample clip of the TSM model, including loading the frames, data transformation, model inference, and displaying the prediction in graphical user interface (GUI). Estimated values are marked with an asterisk (*). The total latency time is around 32.4 ms, which is lower than the 33.3 ms required for a 30 FPS system.

Table 5-7: Latency times for each stage, measured on demo workstation.

Stage	**Latency**
Load frame to buffer	0.5 ms*
Data transformation	15.6 ms
Model inference	15.8 ms
Misc. drawing and display	0.5 ms*
Total	**32.4 ms***

Some sample predictions from the live demonstration were recorded and are shown in Figure 5-5 and Figure 5-6, where the system runs at 29-31 FPS. The images on the left show a side view of the car cabin setup, indicating that the recorded video was taken during live testing. The images on the right show the Kinect RGB Stream from a front view location. Text is printed on the bottom left corner to provide information about the system, while the top left corner displays the top 5 predictions with confidence scores greater than 20%. Low confidence predictions are intentionally omitted to avoid cluttering the GUI.

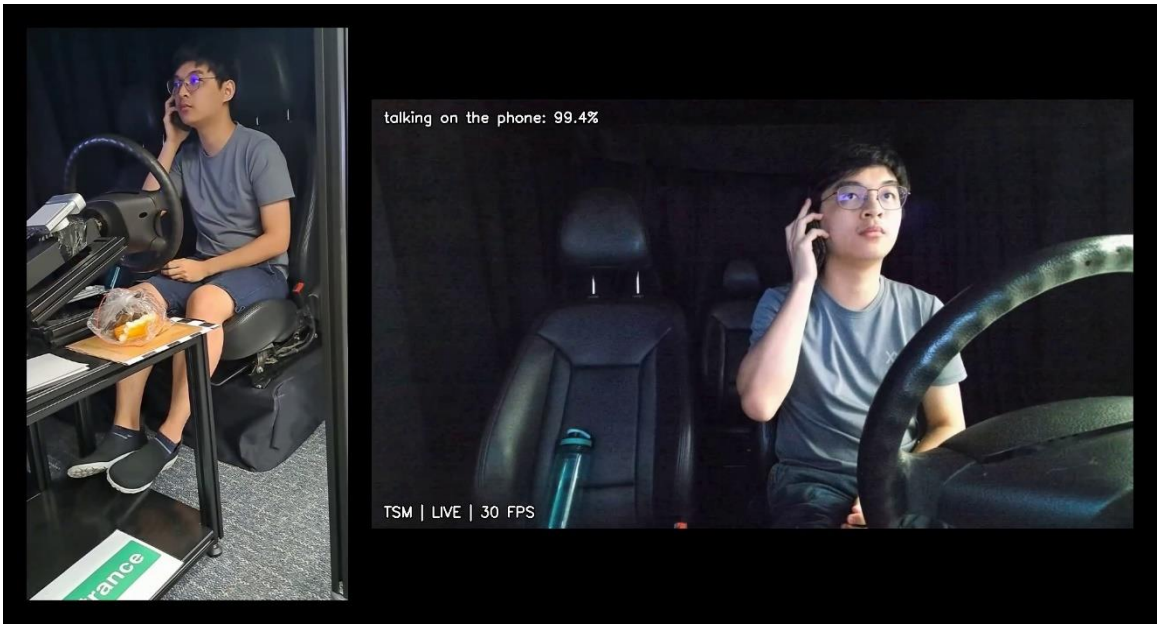

Figure 5-5: Sample frame taken from live demonstration with ground truth “talking on the phone” activity – side view (left), front view (right).

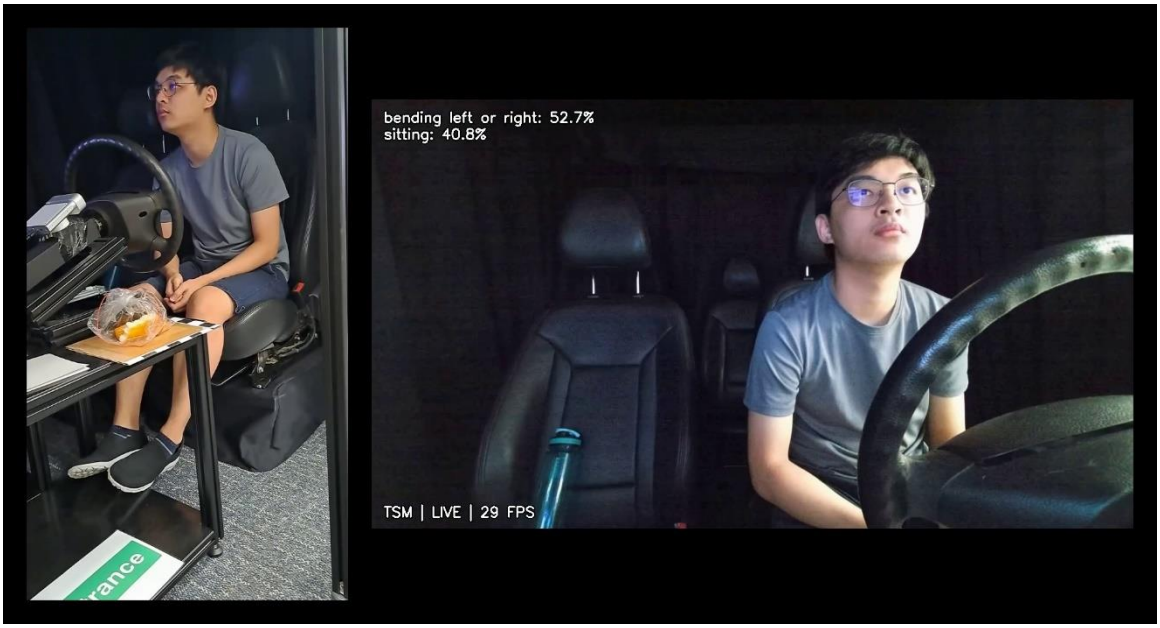

Figure 5-6: Sample frame taken from live demonstration with ground truth “sitting still” activity – side view (left), front view (right).

From Figure 5-5, it can be observed that the TSM model correctly predicted the “talking on the phone” class, with most object interactions predicted correctly, such as “eating”,

“taking a drink’ etc. As for Figure 5-6, the model predicted the activity as “bending left or right” instead of the ground truth label of “sitting still”. However, it is possible that this is a dataset issue since there is not much distinction between actual video samples of these two classes. It should be noted that the “bending left or right” class was provided to depict a theoretical situation where the driver loses consciousness and lays back in the seat.

To conclude, this live demonstration proves that a real-time monitoring system using TSM models is capable of processing video data in real-time with low latency and high accuracy.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

The main objective of this project was to train and develop stable and robust deep learning-based models capable of recognizing driver actions in a car cabin environment. Multiple video action recognition models were adopted and extended to classify human actions in this domain.

As many recorded datasets are not perfect, class imbalance issues often arise simply due to the infrequent nature of rare classes in real life. As such, different training strategies were investigated to enhance the model's robustness to various actions. Modifying the cross-entropy loss to include class weighting and hard sample mining resulted in it being the best technique with an 8-point increase in balanced accuracy compared to the baseline.

Through investigations and experimentation of different models, one needs to balance the tradeoff between the model's accuracy and efficiency. Although the Video Swin Transformer (Base variant) model achieved the highest balanced accuracy score at 70.06, it was six times slower in latency compared to the most efficient model. Therefore, the Temporal Shift Module model was recommended as the most appropriate choice for real-time driver action recognition due to its balanced performance at 68.47 balanced accuracy score and 15.0 ms latency.

While the objective of the project has been achieved at this point, further work was undertaken to increase the reliability and robustness of the models. Multi-modal fusion was performed at both early and late stages of the model, resulting in the best-performing architecture that improved the balanced accuracy score by an additional 5 points compared

to using a single modality stream only.

Furthermore, a real-time prototype was also deployed to showcase a live demonstration using a Temporal Shift Module-based model which was fine-tuned on an internal dataset. The real-time prototype provided more concrete evidence that the application of this model is not only effective in achieving high accuracy on an open-source dataset but also performs well in real-life testing scenarios.

In conclusion, Temporal Shift Module-based models are well-suited for a real-time driver action recognition system as they offer good accuracy and low latency. The technology may be readily incorporated after further testing to ensure that infrequent actions can be recognized with an acceptable detection rate.

6.2 Recommendations for Future Work

There are several directions in which this project's work can be improved upon. Firstly, collecting a more extensive and diverse dataset based on real-world driving scenarios can enhance the model's ability to perform well in unseen cases. Moreover, exploring advanced multi-modal fusion techniques, particularly attention-based methods, which have demonstrated promising results, may also lead to interesting results. Additionally, with the rapidly evolving field of computer vision research, newer and improved model architectures can be adopted and further investigated for this particular use case. Finally, since the current model deployment is on a workstation, further work can be done to deploy the model in a smaller form factor, such as a Jetson NX device.

Reflection on Learning Outcome Attainment

The project has provided me with a comprehensive learning experience, particularly in the following aspects:

- Engineering Knowledge
  - This project aligns closely with my specialization in Data Intelligence and Processing, as it has allowed me to apply my knowledge of deep learning and computer vision to provide solutions to an open problem. Throughout this project, I have gained a comprehensive understanding of various domains and explored cutting-edge topics in the field.

- Design/Development of Solutions
  - This project encouraged me to think outside the box and design a solution suitable for the specific domain of car cabin environments. The solution went through several design iterations to continuously explore and propose new techniques to further improve the model's robustness. A live system prototype was also developed, which provides a practical implementation of the proposed solution.

- The Engineer and Society
  - The motivation behind this project is to increase road safety and reduce instances of distracted driving, benefiting both drivers and other road users, including pedestrians. The deployment of the developed model serves as a promising introduction to a potential system that can achieve this goal, and the detection outputs from the monitoring system can be used for further developments.

References

[1] "Distracted Driving | NHTSA." https://www.nhtsa.gov/risky-driving/distracted-driving (accessed Sep. 10, 2022).

[2] National Center for Statistics and Analysis, *Distracted Driving 2019*, (Research Note. Report No. DOT HS 813 111). National Highway Traffic Safety Administration, 2021.

[3] "SAE International Releases Updated Visual Chart for Its 'Levels of Driving Automation' Standard for Self-Driving Vehicles." https://www.sae.org/site/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles (accessed Mar. 24, 2023).

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1026–1034. doi: 10.1109/ICCV.2015.123.

[5] "What are Convolutional Neural Networks?," Jan. 06, 2021. https://www.ibm.com/cloud/learn/convolutional-neural-networks (accessed Nov. 13, 2022).

[6] A. Rosebrock, "Convolutional Neural Networks (CNNs) and Layer Types," *PyImageSearch*, May 14, 2021. https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/ (accessed Nov. 13, 2022).

[7] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning Spatiotemporal Features with 3D Convolutional Networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: IEEE, Dec. 2015, pp. 4489–4497. doi: 10.1109/ICCV.2015.510.

[8] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge.” arXiv, Jan. 29, 2015. doi: 10.48550/arXiv.1409.0575.

[9] Y. Zhu *et al.*, “A Comprehensive Study of Deep Video Action Recognition.” arXiv, Dec. 11, 2020. doi: 10.48550/arXiv.2012.06567.

[10] W. Kay *et al.*, “The Kinetics Human Action Video Dataset.” arXiv, May 19, 2017. doi: 10.48550/arXiv.1705.06950.

[11] J. Carreira and A. Zisserman, “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 4724–4733. doi: 10.1109/CVPR.2017.502.

[12] L. Wang *et al.*, “Temporal Segment Networks: Towards Good Practices for Deep Action Recognition.” arXiv, Aug. 02, 2016. doi: 10.48550/arXiv.1608.00859.

[13] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.” arXiv, Jun. 03, 2021. doi: 10.48550/arXiv.2010.11929.

[14] Z. Liu *et al.*, “Video Swin Transformer,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 3192–3201. doi: 10.1109/CVPR52688.2022.00320.

[15] Z. Liu *et al.*, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, pp. 9992–10002. doi: 10.1109/ICCV48922.2021.00986.

[16] S. Z. Li, R. Chu, S. Liao, and L. Zhang, “Illumination Invariant Face Recognition Using Near-Infrared Images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 4, pp. 627–639, Apr. 2007, doi: 10.1109/TPAMI.2007.1014.

[17] M. Martin *et al.*, “Drive&Act: A Multi-Modal Dataset for Fine-Grained Driver Behavior Recognition in Autonomous Vehicles,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 2801–2810. doi: 10.1109/ICCV.2019.00289.

[18] J. Lin, C. Gan, and S. Han, “TSM: Temporal Shift Module for Efficient Video Understanding,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 7082–7092. doi: 10.1109/ICCV.2019.00718.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” arXiv, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385.

[20] L. Biewald, “Experiment Tracking with Weights and Biases.” 2020. [Online]. Available: https://www.wandb.com/

[21] P. Goyal *et al.*, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.” arXiv, Apr. 30, 2018. doi: 10.48550/arXiv.1706.02677.

[22] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization.” arXiv, Jan. 04, 2019. doi: 10.48550/arXiv.1711.05101.

[23] MMAction2 Contributors, “OpenMMLab’s Next Generation Video Understanding Toolbox and Benchmark.” Jul. 2020. [Online]. Available: https://github.com/open-mmlab/mmaction2

[24] K. Peng, A. Roitberg, K. Yang, J. Zhang, and R. Stiefelhagen, “TransDARC: Transformer-based Driver Activity Recognition with Latent Space Feature Calibration.” arXiv, Jul. 28, 2022. doi: 10.48550/arXiv.2203.00927.

[25] S. Y. Boulahia, A. Amamra, M. R. Madi, and S. Daikh, “Early, intermediate and late fusion strategies for robust deep learning-based multimodal action recognition,” *Machine Vision and Applications*, vol. 32, no. 6, p. 121, Sep. 2021, doi: 10.1007/s00138-021-01249-8.

[26] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256. Accessed: Mar. 20, 2023. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

Appendix

Table A-1 presents the intermediate results obtained when the model was trained using class weighting only, as compared to the CW+HSM approach used in Table 5-5. It can be observed that the results obtained using CW+HSM were consistently higher, which is why they were selected as the final results.

Table A-1: Summary of proposed multi-modal fusion methods while varying modality combinations, TSM model is trained using class weighting only.

Fusion Method	**Modality**	**Val ↑**	**Test ↑**
No Fusion (single modality)	K_c	68.50	62.58
	K_i	69.28	60.71
	K_d	67.43	58.03
Early Fusion (concatenate channels)	$K_c + K_i$	70.31	62.27
	$K_c + K_d$	64.67	59.93
	$K_i + K_d$	67.26	60.33
	$K_c + K_i + K_d$	67.14	61.32
Late Fusion: Type A (average probabilities)	$K_c + K_i$	71.34	65.03
	$K_c + K_d$	71.49	64.12
	$K_i + K_d$	71.43	63.45
	$K_c + K_i + K_d$	72.73	65.43
Late Fusion: Type B (concatenate scores - MLP)	$K_c + K_i$	70.24	64.40
	$K_c + K_d$	67.89	64.10
	$K_i + K_d$	70.74	61.97
	$K_c + K_i + K_d$	70.41	62.86
Late Fusion: Type C (linear weighted scores)	$K_c + K_i$	70.59	64.23
	$K_c + K_d$	**73.26**	65.35
	$K_i + K_d$	72.27	62.02
	$K_c + K_i + K_d$	73.12	**65.72**