Sign Language Recognition Application by Using Machine Learning Methods

UNIVERSITY OF
LINCOLN

Niracha Chaiwong
26683869

26683869@lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of MSc Computer Science

Supervisor Dr. Miao Yu

7 September 2023

Acknowledgements

I would like to express sincere gratitude to Dr. Miao Yu, my project supervisor, for their invaluable guidance, consistent support, and expert insights throughout the duration of this project. Dr. Miao Yu is an excellent mentor who provides valuable advice about challenges and technical issues during development. This support leads me to the success of this project.

I would also like to extend my appreciation to the faculty members, including module coordinators and staff, who answered my questions during the report writing and encouragement at various stages of this project.

Moreover, I am grateful to the School of Computer Science, University of Lincoln, which provides laboratory support for high-speed internet, great computers and working space. These facilities accommodate me to do better work quality.

Furthermore, I am deeply thankful to my family and friends for their constant encouragement, understanding, and patience during this academic journey. Your company helped me to get through the stresses.

Lastly, I would like to express my gratitude to the academic community for their valuable research in the field of sign language recognition, which provided me with a lot of knowledge and motivation for this project.

This project would not have been successful without the support and collaboration of all those mentioned above. Thank you for being an integral part of this research.

Abstract

Nowadays, many people with hearing impairments in our society use sign language as a communication tool. This project aims to bridge the communication gap between sign language users and those unfamiliar with it by developing a sign language recognition application. This project employs machine learning methods, specifically the LRCN model and 3DCNN model. The baselines of these two methods were proposed and modified to improve the model to achieve the project objective. Firstly, the sliding window method was used to expand the dataset. Moreover, dropout layers and L2 regularisation were applied to reduce the overfitting. As a result, these enhancements significantly improve the model accuracy. Among the challenges addressed by many researchers was training a robust sign language recognition required a large dataset with various environments. This project utilises the WLASL dataset, which is known for its extensive vocabulary and diverse signers. For the implementation, 10 classes were selected. A comparative analysis was also discussed among LRCN and 3DCNN by f1-score for in-class evaluation and weighted average of f1-score for both models. This method ensures that the model performs well to achieve the research objective with results above 80% in each class and 95% and 94% in both models, respectively. Additionally, resource usage was monitored, and showing that LRCN requires more memory than 3DCNN as it needs more epochs to get high accuracy. Lastly, the model was deployed into a web application to test with real-world data, but real-world environments' complexity reduces the model performance. This challenge leads to further research questions for future work. However, the project can be the prototype and initialised the idea of a sign language recognition task.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Summary

Effective communication is a fundamental right that should be accessible to all people, including the deft community, who use sign language as a medium to communicate. Despite the fact that millions of people in the world have used sign language, there is a communication gap between sign language users and people who are unfamiliar with this kind of expression. To reduce the gap between sign and non-sign users, this study utilises machine learning methods to perform sign language recognition by implementing the LRCN model and comparing it with the 3DCNN model to evaluate the best performance on sign language gesture recognition.

1.1 Background

Nowadays, hearing impairment is one of the fourth most common disability in the world. According to the World Health Organization (WHO), the global population is currently experiencing hearing impairments of over 1.5 billion people and is expected to exceed 2.5 billion in the next 30 years (WHO, 2019). One of the essential tools used as a medium to communicate among the deaf community or those who have hearing impairments is sign language. Visual space is used to identify the specific meanings instead of using oral language or sound. Sign language gestures do not use only hand and arm movements but also include facial expressions and various body movements, which mostly focus on the upper part of the body or above the waist level. There are more than 300 sign languages

that have been used in each part of the world to facilitate communication for individuals. American Sign Language (ASL) is considered to be one of the most widely used sign languages for communication in the United States and English-speaking regions in Canada (Kusters, 2021). Since 1993, research in the field of human gesture has been studied, which developed from handwriting to gesture recognition system by detecting, classifying and translating those gestures into words or expressions. Gesture recognition tasks can be divided into two approaches, which are vision-based and sensor-based methods. In vision-based, cameras or investigating techniques have been used, such as body markers, hand location or finger key points. In contrast, sensor-based is focused on embedded technology to capture interested motion and position. Sign language recognition tasks can be divided into two types, which are static and dynamic gestures. The static is a still image representing a word in one frame—for example, finger alphabet spelling. In which, dynamic gestures contain movements of hands or body. Most of them are videos or multiple sequence frames (Cheok, Omar and Jaward, 2019). Sign language recognition is considered a Human-Computer Interaction (HCI) system, which is combined with various techniques such as computer vision, pattern matching, linguistics and natural language processing to translate the meanings (Wadhawan and P. Kumar, 2021). At present, there are technologies that mainly support speaking and writing, such as speech-to-word in What's App, but rarely are applications that support sign language users (Rastgoo, Kiani and Escalera, 2021). Generally, sign language is not widely understood by people who are not familiar with it, and only a few people have the ability to use it. The problem creates a barrier between the two societies, which inspired this project to develop the sign language recognition application by using machine learning methods that have the capability to deal with feature extraction and sequence processing as this project aims to translate gestures into a word. The dataset that will be used is the Word-Level American Sign Language (WLASL) video, which is new and claimed to be one of the biggest datasets in terms of vocabulary and various signers designed especially for developing sign language recognition systems (Li et al., 2020). Performing machine learning techniques with the WLASL dataset is expected to give satisfied model performance and be robust to changes in real-world environments.

1.2 Aim and Objectives

1.2.1 High-level Aims

The main aim of the project is to develop a sign language recognition system by using machine learning that accurately recognises sign language gestures. Moreover, test this model with real-world data by creating an application prototype to reduce the gap between sign language users and those who are unfamiliar with them.

1.2.2 Objectives

To accomplish the above aims, the project is driven by the following detailed objectives, which are divided into two parts: Machine Learning Model and Web Application.

Machine Learning Model

- Develop a robust sign language recognition system by designing and implementing machine learning architectures
- Enhance model performance by train and optimise the model with appropriate hyperparameters and optimise resource usage
- Evaluate the model performance with the test dataset by measuring performance and confusion metrics
- Analyse the strengths and limitations of two machine learning models.

Web Application

- Design a prototype web application with an integrated machine learning model
- Evaluate the performance with the real-world applicability
- Identify challenges and future research directions to improve the project

1.3 Report Structure

This report consists of 7 main parts, which are an introduction, literature review, methodology, software development, results, discussion and evaluation, and conclusion and future work. The details can be found as follows:

Introduction: This section provides a comprehensive introduction to the core concepts of the project and presents an overview of the project's fundamental concepts, including its aims and objectives. Additionally, introduces background information and outlines the structure of each section in this report.

Literature review: This section summarises existing research on sign language recognition, machine learning methods, and related works. Moreover, it identifies gaps, challenges, and opportunities in this research area, including a discussion about the methodologies and algorithms that are commonly used in sign language recognition. Lastly, identify the direction of this project, the methods used to perform the experiment, and the software solution.

Methodology: This section details the project management method of this project, describes the dataset used, data preprocessing and feature extraction techniques. Moreover, explains the implementation of models for sign language recognition and outlines the training process, hyperparameter tuning, and evaluation metrics used.

Software Development: This section presents the technical details of the sign language recognition application (tools and machine environments), discusses the software architecture, and addresses the challenges and limitations of the model until testing and deploying the model.

Results: This section presents the output and results of the experiments and evaluations, including performance metrics.

Discussion and Evaluation: This section discusses and interprets the results in the context of the project aims and objectives, analyses the strengths and limitations of each model and discusses the findings of the results.

Conclusion and Future Work: This section summarises the key contributions of the project research. Moreover, proposes ideas for future research and improvements to this project.

Chapter 2

Literature Review

Summary

In the past years, machine learning has been used to assist and facilitate humans in our society. This research will focus on developing a sign language recognition system by using the advantages of machine learning to make a robust and practical application. To achieve the project aim and objective, various research and studies have been reviewed in terms of related work and neural network architecture techniques.

2.1 Related Work

There are several research that focused on using machine learning methods to predict and translate human gestures. Survey research in the field of intelligent tools for sign language recognition has been reviewed and states that in dealing with input videos, both feature extraction and sequence processing need to be considered. For the feature extraction approach, CNN has been widely used along with other methods for sequence processing, such as GRU, RNN, and LSTM. Additionally, two or more models have been combined to enhance the model's performance. For the input data, several structures have been used to train the model, such as RGB videos, skeleton, thermal, depth, and flow of information (Rastgoo, Kiani and Escalera, 2021). CNN and RNN were used to predict a word from self-collected RGB videos. Moreover, data argumentation was used to increase the size of the dataset. The author indicated the challenges of various skin tones, facial features, and clothing affected the model's accuracy (Sabeenian,

Bharathwaj and Aadhil, 2020). CNN and LSTM were used to predict the Bangla Sign Language (BSL), which achieved 88.5% testing accuracy. The author addressed that overfitting is the main problem during the training process (Basnin, Nahar and Hossain, 2020). CNN combined with LSTM and RNN have been compared by using Chinese Sign Language (CSL), in which LSTM performed better than RNN in terms of storing and accessing information (Yang and Zhu, 2017). This method is also used in Indian sign language, where extra LSTM was used to improve the model performance (Aparna and Geetha, 2020). Above 2DCNN with other sequence processing, 3DCNN is also one method that can handle dynamic gesture recognition. This method achieved a satisfactory performance in all scores of precision, recall and f-measure. To detect both hands accurately, CNN can be used to combine with spatial transform layers, which can deal with a multi-label environment (Sharma and K. Kumar, 2021). Moreover, 3DCNN was used in feature extraction for sign language recognition and the prior experience of the model is not required to train the 3DCNN model. It can be used directly to identify the clue (Huang et al., 2018). Other methods besides CNN have also been used, such as 3D hand key points, which were used to predict the real-time sign language in order to find relationships and angles between each finger, and LSTM was used as a temporal sequence learning. The model gave over 99% accuracy for all signs. The challenges addressed were misclassification due to the similarities of some signs, and both hands were difficult to detect accurately (Rastgoo, Kiani and Escalera, 2022). Transformer method was used for the recognition system for continuous sign language translation. The challenges were the limitation of the labels dataset and large amount of information needed to get the optimal model performance (Camgoz et al., 2020).

To conclude and discuss the relevant research, several research studies have been conducted on sign language recognition systems by using machine learning methods such as CNN, 3DCNN, RNN, LSTM and Transformer. Most research focuses on CNN with other methods for sequence processing as the input is video sequences. Various inputs were used to train the model, such as RGB videos, skeleton or hand key points. Some authors discussed the challenges faced during the development process, which are the variations in skin tone, shirt colour, facial structures, and light conditions that drop the model accuracy. Another important challenge is the amount of training data that

required a large amount of information to train the model to be robust to environmental changes. The most common techniques used are data argumentation and self-collected data. From the related work reviewed, this project will integrate the data from the Word-Level American Sign Language (WLASL) dataset, which is claimed to be one of the largest ASL datasets with the LRCN model, which combines both CNN and LSTM together. This project aims to improve model performance through the combination of both components. Furthermore, introduced a prototype of the sign language recognition application.

2.2 Neural Network Architectures and Techniques

In this section, specific neural network architectures and techniques that have been used to develop the sign language recognition system will be introduced and discussed.

2.2.1 Convolutional Neural Networks (CNNs)

The Artificial Neural Network (ANN) is a computer processing system based on the nervous system of human beings. There are numerous interconnected nodes called neurons, which have the ability to learn information from input to solve specific tasks. Methods to train the ANN model can be classified as supervised, unsupervised learning and reinforcement learning. Supervised learning requires the label along with the data, whereas unsupervised learning can learn from unlabeled data, and reinforcement learning learns from trial and error while interacting with the environment (Sathya, Abraham et al., 2013). CNN is a supervised learning method and is one of the best deep-learning tools for image processing and recognition tasks such as face detection, image classification and video recognition. CNN consists of numbers of a convolutional layer, a nonlinearity layer, a pooling layer, and a fully connected layer, which enables the model to capture specific features from the input images, making it appropriate for handling the image data (Albawi, Mohammed and Al-Zawi, 2017). In the past few years, CNN has gained significant interest in the field of vision-based applications for SLR systems, which are able to extract features, downsample, and mitigate the issue of overfitting effectively (Nimisha and Jacob, 2020).

Two-dimensional Convolutional Neural Networks (2DCNN) is an artificial neural network that is designed to perform spatial feature extraction to analyse and interpret input images. It consists of multiple layers in which convolutional, pooling and nonlinear activation operations were applied to the input data in two directions (x and y). This method can be used to extract features and patterns information from the input data and is suitable for classification, segmentation, or detection tasks. Performing 2DCNN in video processing has limitations in its ability to perform sequence processing. The integration of additional methods is needed in order to achieve the objective successfully (Vafeiadis et al., 2022).

Three-dimensional Convolutional Neural Networks (3DCNN) is similar to 2DCNN. Beyond a normal CNN, it applies convolutional, pooling, and nonlinear activation operations to input data in three directions (x, y, and z). This capability makes this model analyse volumetric or temporal data effectively. It enhances the model's ability to learn spatiotemporal features from the input data in depth sequences, which can be used in tasks related to gesture recognition. (Zhang et al., 2017).

2.2.2 Long Short-Term Memory (LSTM) Networks

LSTM is one of the RNN (Recurrent Neural Networks) that has the ability to capture and learn long-term relationships inside sequential prediction tasks effectively. LSTM can deal with the vanishing gradient problem that RNN faced, in which the gradient became too small when training the model. it prevents the model from updating the weights and continuing training. Moreover, The LSTM model employs multiplicative gate units in order to learn when to access the cells and also forget or update the internal states. It uses a Sigmoid function to get output in 0 and 1, then multiplied with the cell state to control the transmission of information, which allows LSTM to keep or forget the information based on the input data. This ability to handle long-term dependencies makes it more effective than other RNNs (Staudemeyer and E. R. Morris, 2019). Furthermore, the LSTM model is widely used to combine with the CNN model and get high performance. After CNN processes feature extraction, LSTM is used to do sequence learning (Rehman et al., 2019).

2.2.3 Long-term Recurrent Convolutional Network (LRCN)

LRCN is one of the neural networks that combine both convolutional layers and recurrent layers to deal with visual and sequential data, which is suitable for video description, activity recognition, and image captioning tasks, where the input has various lengths and temporal dynamics. The LRCN make use of CNN in terms of feature extraction and LSTM to learn sequential patterns from the input data. It is an end-to-end trainable method that acquires spatial and temporal representations without the need for manually modified features. The differences between LRCN and CNN+LSTM are that LRCN is a framework that can be used for tasks involving sequential inputs and outputs, including activity video recognition, whereas CNN+LSTM is a model that is mainly used for image captioning. LRCN sends the visual features extracted by CNN to LSTM at every time step, whereas CNN+LSTM only passes visual features to LSTM during the initial time step and then applies the LSTM to generate a natural language sentence. Moreover, LRCN assesses various LSTM architectures, but CNN+LSTM employs a single-layer LSTM with a straightforward beam search technique (Donahue et al., 2015).

2.2.4 TimeDistributed Layer

The TimeDistributed is one of the Keras functions. This layer acts like a wrapper that allows each layer to apply to every temporal slice of an input. This implies that the temporal relationships between different time steps in sequential data can be preserved. Using TimeDistributed instead of applying a layer to the whole sequence at one time can reduce the computation time and resources required because it reduces the number of parameters, which speeds up the training process. Moreover, it is flexible and can adapt to other different input lengths and shapes. However, there are some limitations to using this method. Firstly, some global dependencies and patterns may not be captured throughout the entire sequence, which can affect the model performance. The complexity in the design and difficulty in debugging the model. Lastly, hyperparameter tuning and regularisation techniques may required to prevent the issue of overfitting or underfitting (Narang and Singh, 2023). To use the TimeDistributed from Keras, each

input must have a minimum of three dimensions, and the dimension at index one of the first input will be regarded as the temporal dimension. For example, if there are 32 video samples with 128x128 RGB images and these video samples extend through a sequence of 10 timesteps, the input shape will be (32, 10, 128, 128, 3). Later on, it can be employed to implement the Conv2D layer uniformly across all timesteps with constant weights used for the Conv2D operation (Team K, n.d.).

2.2.5 Model Enhancements

Sliding Window Method

Sliding Window is one of the data argumentation methods that generate more data samples from existing datasets. It can improve the performance and stability of the model in a limited dataset. This method will capture input data into smaller segments with a fixed length or size. Each segment will be treated as new sample data for the deep learning model. Using a Sliding Window can prevent the model from overfitting. Moreover, it preserves the temporal relationships between each time step in sequential data and allows for parallel processing of the segments, which can speed up the training process (Lashgari, Liang and Maoz, 2020). The Sliding Window technique performs neighbourhood operations on every input image pixel with n x n specified size of the window. It is widely used in various computer vision tasks, such as object detection, image classification, and semantic segmentation (Shorten and Khoshgoftaar, 2019).

Dropout and L2-regularisation Layer

Dropout is one of the regularisation methods that has been widely used to prevent overfitting of the model during the training process by randomly dropping out units during training. Overfitting occurs when the model learns too well from training data but fails to perform in unseen data. Applying the dropout technique will randomly disable neurons and corresponding connections during the training process, which will prevent the model from relying on some specific neurons only but make it learn from all neurons to enhance the model's robustness and generalisation. The number of neurons that will be drooped depends on the parameters set. For example, p = 0.25 means

that one-quarter of neurons from that layer will randomly be drooped. Normally, the parameters will be set in the range of 0.2 to 0.5. Dropout has been used in various tasks, such as image classification, natural language processing, and speech recognition. It can also be applied in the neuron network model, such as feedforward, convolutional, recurrent, and transformer networks. It is one of the most simplest ways to enhance the model performance and reliability of neural networks (Salehin and Kang, 2023). The L2 regularisation also has the ability to reduce overfitting and deal with the stability of the model training process. This method will reduce the magnitude of weights by adding a regularisation term (λ) proportional to the square of the weights, as shown in the equation 2.1. Additionally, L1 regularisation is also one technique that can deal with overfitting issues. However, L2 regularisation provides more effective results than L1 regularisation (Bir and Balas, 2020).

$$L2 = (\frac{1}{2}) \times \lambda \times \|w\|^2 \tag{2.1}$$

Early Stopping

Early stopping is used to prevent the model from becoming too complex and losing the ability to be generalised. It will stop training when the model becomes overfit by setting the number of iterations to run the training. This method has been employed in many machine learning methods. One of the implementations is to split the data into training and validation sets where the model will be learned only on the training set and evaluated in the validation set once per epoch. This method allows us to monitor the validation error. If the error stops improving or starts increasing, the training process will be stopped (Ying, 2019).

Chapter 3

Methodology

Summary

This section presents a comprehensive overview of methodologies and techniques used in this project, which breaks down into project management, including the Gantt chart to outline the tasks with milestones, the dataset used, data preprocessing techniques, model implementation, training processes, hyperparameter tuning, and evaluation metrics. These methods are fundamental components of a sign language recognition system that are essential for the success of this project.

3.1 Project Management

Efficient project management is essential for achieving the project's efficiency within time constraints. The chosen project management methodologies and tools for the successful development of the sign language recognition application will be explored. This includes the project management approach, constructing a Gantt chart for planning, a comprehensive risk assessment, and version control mechanisms for the project.

3.1.1 Project Management Methods

Effective project management is important for project success. So, carefully considering each method's strengths and weaknesses is needed. In this section, the advantages of the selected method and the reasons why alternative methods were not suitable for this project will be discussed.

Waterfall Model is the software development model that follows a linear or sequential approach where tasks must be completed in order without revisiting previous ones. This makes it difficult to revise earlier tasks and can lead to costly modifications for adjustments. (A. A. Adenowo and B. A. Adenowo, 2013). The development team will only focus on the requirements during the planning process, and the feedback will be received after the final product release. The waterfall model can provide a clear overview of the project and its requirements for new developers, but it can be difficult to make changes once it is completed (Singhto and Phakdee, 2016).

Incremental Model is the combination of the iterative and waterfall models by dividing the project into iterations that each contain their own waterfall phases. If the result from one iteration does not meet the user's requirements, the next iteration will build upon the previous one. In the first increment, the basic core requirement will be addressed, while additional functionality will be added in subsequent steps. This model enables users to interact with the project, see the core function earlier, and provide feedback on each increment. However, it requires good planning and design processes and cannot be repeated in each increment (Alshamrani and Bahattab, 2015).

Agile Methodology is a project management methodology that breaks projects into dynamic parts called sprints. It emphasises collaboration and continuous improvement. By adopting this method, we can improve work quality and adapt to changing requirements with flexibility (Malik, Ahmad and Hussain, 2019). It implements techniques and management processes to handle changes that arise during software development, including evolving experiences, shifting requirements, and modifications in the development process (Turk, France and Rumpe, 2002). This method allows us to focus on smaller tasks, which can minimise the risks and probability of any errors or modifications in the requirements (Alfonso and Botia, 2005). In this project, the Agile method has been selected as the primary project management template because of its flexibility for changes and its ability to reduce risks through effective sub-task division. Scaling up the project to accommodate larger train data is also possible and easier. In order to do sprint planning, I have listed the tasks as project planning, software design, software development, evaluation and testing, and deploying the model. These phases are further divided into three iterations (sprints), which are data processing, model archi-

tecture development and deployment to the web application. The example of sprints is shown in figure 3.1, and sprint planning is shown in 3.2. More information will be provided in *section 3.1.2: Gantt Chart*.

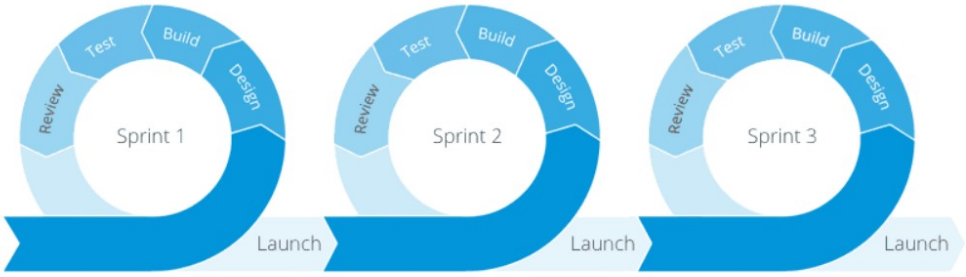

Figure 3.1: Example of sprints (ProductPlan, n.d.).

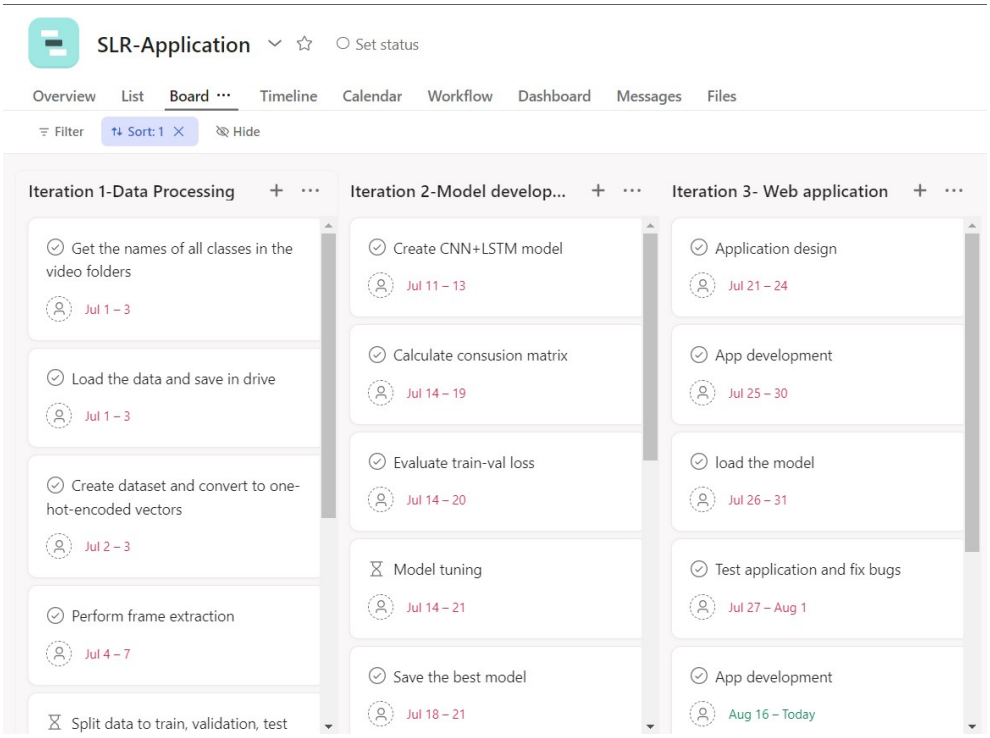

Figure 3.2: Sprints planning for this project

3.1.2 Gantt Chart

A Gantt chart was used to plan and track the project's progress, including task dependencies and milestones. It was regularly updated to reflect any modifications to the timeline. The proposed Gantt chart is shown in figure 3.3. While performing the project, the new Gantt chart was created based on Asana software, where the sprint

management tasks can be constructed. Moreover, keep track of the process and update the finished tasks daily. The sprint tasks and updated Gantt chart are shown in figure 3.2 and 3.4. My project timeline closely followed the proposed Gantt chart due to the flexibility provided by the Agile method. Errors were fixed during each iteration instead of modifying the code from the beginning, which can be time-consuming and complicated.

The main tasks consisted of :

1. Data prepossessing: Load the data, get the label from the folder name, create the dataset as a vector, extract video frames, resize input frames, split data before training and check the data and label if downloaded correctly.

2. Model development: Construct CNN model, apply LSTM model, evaluate model performance, tune the model, save the best model. (The same process was applied to the 3DCNN method).

3. Application development: Design the application, load the model to be deployed and test the application.

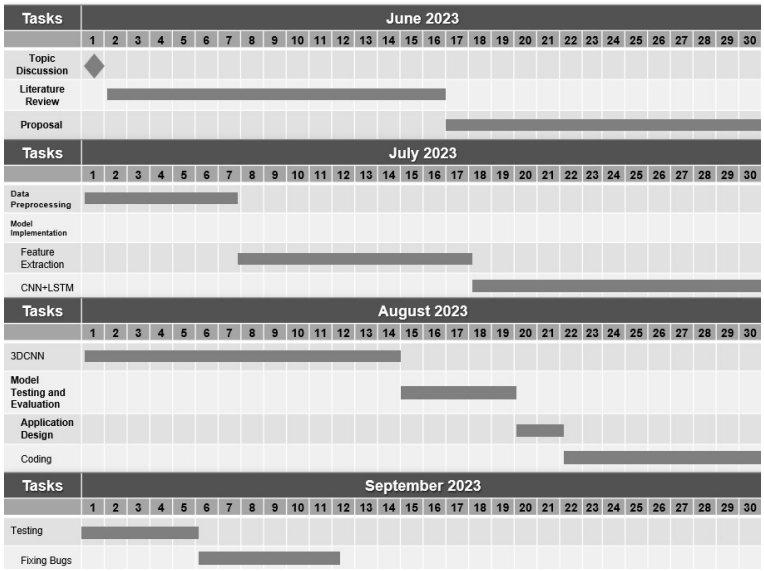

Figure 3.3: Proposed Gantt Chart

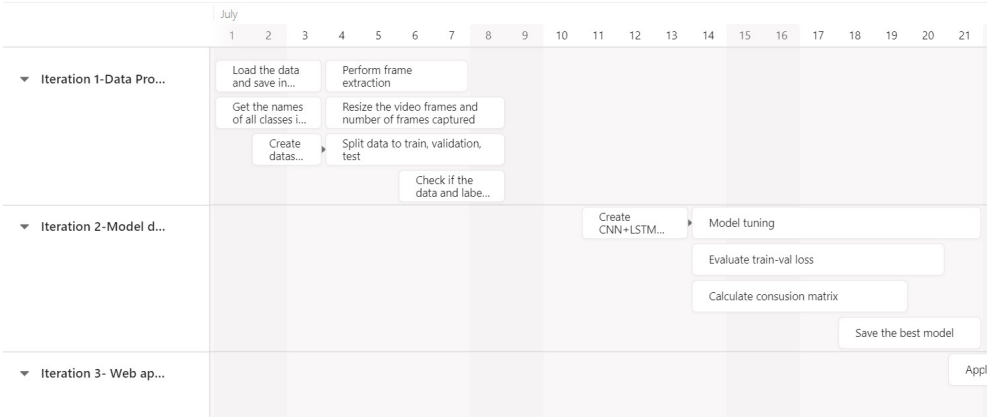

Figure 3.4: Updated Gantt Chart

3.1.3 Risk Assessment

In this section, potential risks that may occur during this project will be identified, along with their potential impacts and strategies for mitigation. It is essential to proactively identify these risks before commencing experimentation to facilitate timely planning and resolution. Some potential risks are identified as shown in table 3.1.

Risk Identification	**Risk Effect**	**Mitigation Strategy**
Data quality and availability	The model cannot learn well from the dataset	Apply data augmentation to increase size of dataset
Construct the model architecture	The model causes overfitting or poor accuracy	Adjust hyperparameters and add layers to prevent overfitting e.g., dropout and batch normalisation
Limited computing resources	Slow down the training process or not enough memory to train	Upgrade to ColabPro for more resources
Limited time to develop the model	The model gives lower performance than the expectation	Adapting the Agile method to achieve maximum efficiency in work

Table 3.1: Risk Assessment and Mitigation Strategy

3.1.4 Version Control

This section describes the version control method used for this project, which was carried out using Google Colab. Even when working in cloud-based settings like Google Colab, version control is essential to managing software development projects. The consistency of the project is preserved, individuals may collaborate more easily, and a precise record of modifications is maintained. Google Colab is a cloud-based service that enables machine learning education and research. It provides a pre-configured runtime environment optimised for deep learning tasks and free access to powerful GPUs with limited usage (Carneiro et al., 2018). Security and Access Control also need to be considered to prevent unauthorised access and modifications. The Colab has this function to add and allow access only to authorised users.

3.2 Software Development Methods

For this project, we employ the Extreme Programming (XP) model to ensure the delivery of software that satisfies the user's requirements or the objective of this project. XP is one of the well-known agile software development methodologies that prioritise user satisfaction throughout time constraints. The development team can rapidly respond to specific changes in requirements. Moreover, users will participate in the process with the team. So, the software can be produced to satisfy criteria, and developers can effectively handle problems (Carroll and D. Morris, 2015). The test cases will be organised into small, manageable iterations such as data processing, model architecture design, and deployment. This approach ensures continuous progress, allows to receive user feedback early and makes it easier to introduce modifications. The XP model is particularly suitable for this project, as presenting work to the supervisor may lead to necessary changes and adjustments. Adopting this model enables us to accommodate modifications and update requirements productively and flexibly.

3.3 Research Methods

This section will provide an overview of the research techniques used to accomplish the project's objectives and provide significant results. These techniques are essential for understanding of how efficiently the sign language recognition software performs.

3.3.1 Experimental Design

The experimental design involves developing and evaluating sign language recognition models developed during the project. The overall process is shown in figure 3.5.
Which includes:

1. Data Processing: Perform data processing to ensure that the data that feed into the model are clean, correct and suitable for training and testing the model
2. Model Training: Construct and employ the machine learning models, including LSTM and 3DCNN, to train the dataset from the previous step
3. Model Tuning: Perform hyperparameters tuning to get the optimise model performance and save the best model
4. Evaluation Metrics: Calculate confusion matrix, accuracy, precision, recall, and F1-score to observe the model performance
5. Result Analysis: Interpret the experiment results and limitations
6. Deploy the model: Deploy the best model to make the web application

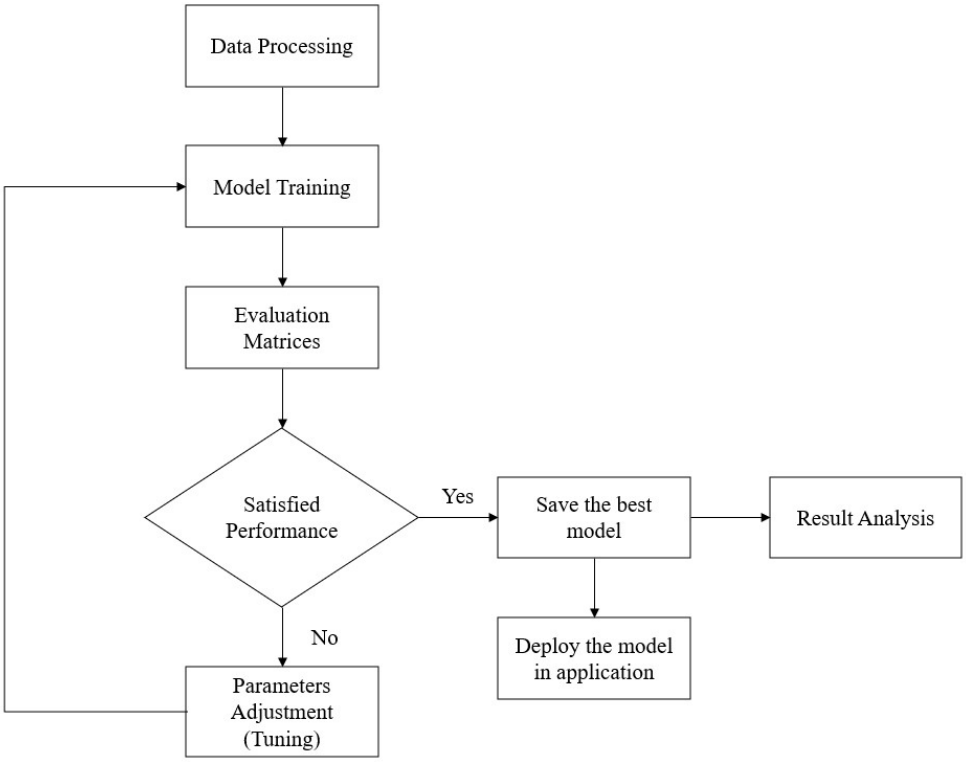

Figure 3.5: Overall Experimental Design Flowchart

3.3.2 Software Requirements

Developing a sign language recognition program requires precise and efficient software requirements. The project is built on top of these specifications.

1. Dataset: The dataset used in this project is the Word-Level American Sign Language (WLASL) video dataset, which is claimed to be the largest public dataset in terms of vocabulary and number of samples in each class. It consists of useful daily words with various signers. This dataset contains 2,000 words, 21,083 videos and 119 signers. In each sample, the average video is 10.5 samples per word. The dataset available at https://dxli94.github.io/WLASL/ (Li et al., 2020). Due to limited resources and time, only 10 classes were used to demonstrate the model prototype in this project.

2. Python: Python is an object-oriented, high-level programming language with dynamic semantics. It has built-in data structures, dynamic typing, and dynamic binding, making it popular for Rapid Application Development. The Python version used in Google Colab is version 3.10.12 (Python, 2019).

3. TensorFlow: TensorFlow is the open source for the machine learning approach which used to construct LRCN and 3DCNN model in this project. The version used is 2.12.0 (TensorFlow, 2019).

4. Keras: Keras is one part of the TensorFlow, which is used to design and configure neural networks. It can provide a high-level, user-friendly interface for creating effective neural architectures. The version used is 2.12.0 (Keras, 2019).

5. NumPy: NumPy is used to perform efficient numerical calculations and data processing in the training process, which is the fundamental package in the Python library. The version used is 1.23.5 (NumPy, 2009).

6. Scikit-learn (sklearn): Sklearn is used for many machine learning approaches, which are data processing and model performance evaluation. The version used is 1.2.2 (scikit-learn, 2019).

7. Matplotlib: Matlopt is used to create data visualisation and graphs after training the model to see the model performance and during model evaluation to present the result. The version used is 3.7.1 (Matplotlib, 2012).

8. Streamlit: Streamlit is the framework used for creating the application's user interface. It is one of the flexible Python modules to deploy machine learning models that are easily integrated with interactive web applications. The version used is 1.25.0 (Streamlit, n.d.).

3.3.3 Software Design Documentation

The success of the project depends on the software solution's practical design. This section emphasises the importance of structure design that is based on established software requirements. As highlighted in the literature review section, CNN and LSTM methods have demonstrated superior performance in feature extraction and sequence processing, which are directly relevant to our project focused on video recognition. This project will conduct a comparative analysis between two approaches for sign language recognition, which are CNN combined with LSTM and 3DCNN. Both methods have been widely studied and applied in the field of video recognition.

3.3.3.1. Data Processing Structure

To achieve the objective of this project, one of the most important parts is data selection. Good data leads the model to learn efficiently and give a satisfying performance. Since this project will be working on sign language, there are various data available to use. The criteria for selecting data are the reliability of source correctness. After selecting the data that will be used, data processing needs to be performed before feeding into the model. This includes grouping, frame extraction, data argumentation and splitting data for the model to learn best and achieve the highest model performance. This research selected 10 classes to demonstrate the model, including "deaf", "help", "no", "thin", "walk", "yes", "finish", "hot", "many"and "bad".

1. Data Grouping: Since the raw data may come in an unorganised file, the data need to be grouped into the correct category folders and labels. There are more than 10,000 videos that can not be processed by hand. Most datasets provide the JSON file containing video IDs and labels. Data grouping can be done by using Python to read the raw videos and JSON files, and then create folders for the gloss in the output directory.

2. Frames Extraction: As the input is video, it needs to be captured into sequences of each sign language gesture and transformed into a series of individual frames, which will be used to construct a temporal sequence. The frames will be resized into 64 X 64 heights, and widths to reduce the calculation time, and the number of frames that feed to the model is 50 as one sequence. The plot of frames captured is shown in figure 3.6.

3. Data Argumentation: Since the available dataset for sign language might be limited and not enough for the model to learn, data argumentation is needed to increase the input data for the model to learn from them. A large amount of data can make the model find the hidden information, robust to changes and give higher performance. However, large data leads to more calculation time, and limited resources also need to be considered.

4. Create Dataset: Create a dataset function to store features and label them, then convert them into one-hot-encoded vectors using Keras's to_categorical method to convert category data into an integer.

5. Split Data: The data has been divided into the ratio of 75:25 by using train_test _split from sklearn as shown in figure 3.7. 75% for training the model and the rest for testing and validation. The testing dataset is a separate dataset to test the generalisation of the model to see if the model can work well with unseen or real-world data. It can be used to observe the overfitting when the model performs well on the training set but performs poorly on the unseen dataset. The validation is used for the tuning process, which is used to select the hyperparameters that give the best model performance, such as epoch, batch size, and learning rate.

Figure 3.6: Frames Extraction

```
# Split the Data into Train ( 75% ) and Test Set + Validation set ( 25% ).
features_train, features_temp, labels_train, labels_temp = train_test_split(
    features, one_hot_encoded_labels, test_size=0.25, shuffle=True, random_state=seed_constant)
features_val, features_test, labels_val, labels_test = train_test_split(
    features_temp, labels_temp, test_size=0.5, shuffle=True, random_state=seed_constant)
```

```
Train Features shape: (2009, 50, 64, 64, 3)
Train Labels shape: (2009, 16)
Validation Features shape: (335, 50, 64, 64, 3)
Validation Labels shape: (335, 16)
Test Features shape: (335, 50, 64, 64, 3)
Test Labels shape: (335, 16)
```

Figure 3.7: Data Splitting using Sklearn

3.3.3.2. LRCN Model Architecture

The Long-term Recurrent Convolutional Network (LRCN), method is a combination of Convolutional Neural Networks (CNN) and Long Short-Term Memory Networks (LSTM) with TimeDistributed layers. Through the utilisation of TimeDistributed layers, it wraps the input data and enables it to be processed as a single shot. This method is designed to take advantage of both the spatial and temporal information contained in sign language gestures and allow accurate recognition. The model structure is shown in figure 3.8, which feeds the captured frames into the CNN model for extracting features and passes to the LSTM mode for processing frame sequences. The Baseline of LRCN model is shown in 3.9.

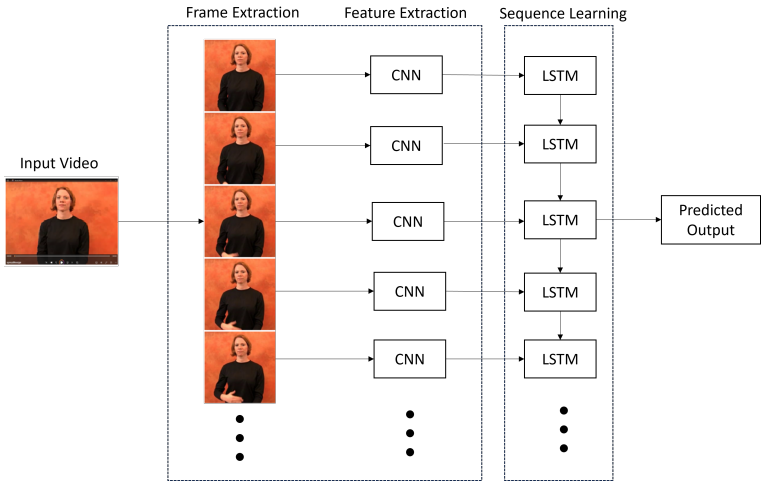

Figure 3.8: LRCN Model

Baseline of LRCN Model Structure:

1. Convolutional Layer (Conv2D): This layer will perform the convolution between input and filters or kernels to capture spatial data. In this layer, filter size (3x3) and 16, 32, 64 number of filters have been applied. Moreover, ReLU (activation='relu') has been used as an activation function to introduce non-linearity.

2. Pooling Layer (MaxPooling2D): A pooling layer is used to reduce the dimensions of feature maps by taking the maximum value. Pooling filter sizes 4X4 and 2X2 have been applied after the convolution layer.

3. Flatten Layer: The Flatten layer is used to transform the 2D array result into a 1D array before feeding into the fully connected layer.

4. Dense Layer: A Dense layer is a fully connected layer that uses the Softmax activation function to calculate probability distribution over multiple classes based on the features learned by the previous layers.

5. TimeDistributed Layer: The TimeDistributed layer enables each layer to process sequence data in each time step of a sequence, including Conv2D, Maxpooling2D, and Flatten layer.

6. LSTM (Long Short-Term Memory) Layer: LSTM is one of the recurrent layers used for sequence modelling. LSTM provide internal memory to recall information from previous time steps that give the model the ability to learn complex temporal patterns. It can determined by the number of memory cells used, which is 32 as the baseline.

```
# Define the 2DCNN+LSTM Model Architecture.
#############################################################################################################

model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu'),
                          input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

model.add(TimeDistributed(MaxPooling2D((4, 4))))

model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((4, 4))))

model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same', activation='relu'), name='custom_conv2'))

model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Flatten()))

model.add(LSTM(32)) #number of memory cells or units in the Long Short-Term Memory (LSTM) layer.

model.add(Dense(len(CLASSES_LIST), activation = 'softmax'))
```

Figure 3.9: Baseline LRCN Model

3.3.3.3. 3DCNN Model Architecture

The other practical method that may effectively classify videos according to their temporal features is 3D-CNN. This model replaces 2D-CNN with 3D filters and pooling kernels, which consider height, width, and time (or depth). So, the LSTM or any other Recurrent Neural Network is not necessary. Also, the 3D model performed better in terms of accuracy and sensitivity than 2D-CNN and reduced false-positive (Yu et al., 2020). The model structure is shown in 3.10. The baseline of 3DCNN is similar to 2DCNN, as shown in figure 3.11.

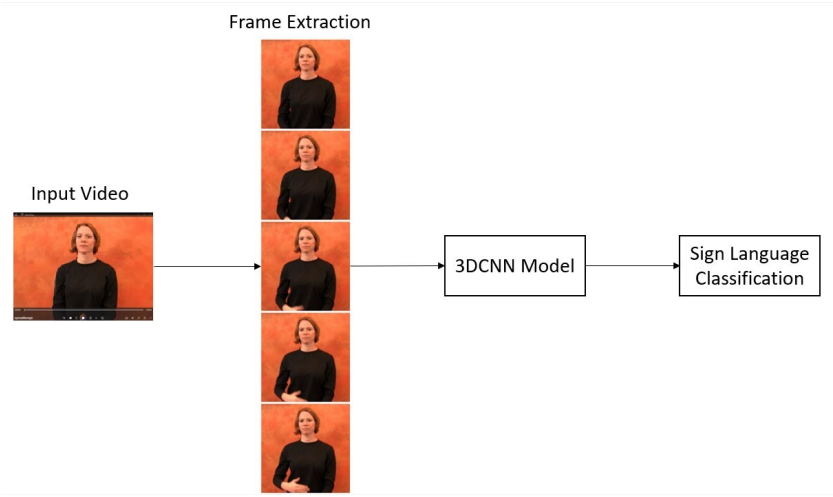

Figure 3.10: 3DCNN Model

Baseline of 3DCNN Model Structure:

1. Convolutional Layer (Conv3D): This layer will perform three-dimensional convolutions between input and filters or kernels to capture spatial data. In this four-layer, filter size (3x3X3) and 16, 32, 64 number of filters have been applied. Moreover, ReLU (activation='relu') has been used as an activation function to introduce non-linearity.

2. Pooling Layer (MaxPooling3D): A pooling layer is used to reduce the dimensions of feature maps by taking the maximum value. Three-dimensional pooling filter sizes 1X4X4 and 1X2X2 have been applied after the convolution layer.

3. Flatten Layer: The Flatten layer is used to transform the multi-dimensional array result into a 1D array before feeding into the fully connected layer.

4. Dense Layer: A Dense layer is a fully connected layer that uses the Softmax activation function to calculate probability distribution over multiple classes based on the features learned by the previous layers.

```
# Create a Sequential model
model = Sequential()

# Define the Model Architecture
########################################################################

model.add(Conv3D(16, (3, 3, 3), padding='same', activation='relu',
                 input_shape=(SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

model.add(MaxPooling3D((1, 4, 4)))

model.add(Conv3D(32, (3, 3, 3), padding='same', activation='relu'))
model.add(MaxPooling3D((1, 4, 4)))

model.add(Conv3D(64, (3, 3, 3), padding='same', activation='relu'))
model.add(MaxPooling3D((1, 2, 2)))

model.add(Conv3D(64, (3, 3, 3), padding='same', activation='relu'))
model.add(MaxPooling3D((1, 2, 2)))

model.add(Flatten())

model.add(Dense(len(CLASSES_LIST), activation='softmax'))
```

Figure 3.11: Baseline 3DCNN Model

3.3.3.4. Compilation and Training Model

Once the machine learning models were constructed, the process of compiling and training the models was performed as shown in figure 3.12.

1. Early Stopping Callback: Early stopping was implemented to prevent overfitting by monitoring validation loss during training and stopping when the loss stops improving from a specified value (patience).

2. Compilation: Categorical cross-entropy has been used as a loss function since it is generally used in multi-class classification tasks. Adam's method has been used as an optimiser for stochastic gradient descent, and accuracy has been used to monitor the training performance.

3. Model Training: Training data has been used to train the model from a small value of epoch and batch size. Then, the model performance will be compared with validation data during the training process. The parameters will be adjusted based on this step.

```
# Create an Instance of Early Stopping Callback.
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min', restore_best_weights = True)

# Compile the model and specify loss function, optimizer and metrics to the model.
CNN3D_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'], run_eagerly=True)


# Start training the model.
CNN3D_model_training_history = CNN3D_model.fit(x=features_train, y=labels_train, epochs=50, batch_size=16,
                                               shuffle=True, validation_data=(features_val, labels_val),
                                               callbacks=[early_stopping_callback])
```

Figure 3.12: Compile and Model Training

3.3.3.5. Performance Evaluation

1. Loss and Accuracy: Training and validation loss and accuracy have been plotted to visualise the training and validation loss and accuracy curves over epochs. To see if there is an overfitting and how to adjust the hyperparameters.

```
import matplotlib.pyplot as plt

# Plot the training, validation loss and accuracy
train_loss = CNN3D_model_training_history.history['loss']
val_loss = CNN3D_model_training_history.history['val_loss']
epochs = range(1, len(train_loss) + 1)
train_accuracy = CNN3D_model_training_history.history['accuracy']
val_accuracy = CNN3D_model_training_history.history['val_accuracy']
epochs = range(1, len(train_accuracy) + 1)
```

Figure 3.13: Training, Validation Loss and Accuracy Plot

2. Classification Report: Precision, Recall, F1-Score, and Support has been used to provide detailed insights into the model performance beyond accuracy. Moreover, Support values can tell the actual number of occurrences of each class.

3. Confusion Matrix: The confusion matrix is a useful tool to evaluate the classification performance of a model that provides information about true positives, true negatives, false positives, and false negatives.

```
# Get the classification report for the training data
classification_report_train = classification_report(labels_train.argmax(axis=1), predicted_labels_train_onehot, target_names=CLASSES_LIST)

# Print the classification report
print("[INFO] evaluation testing for training data ...\n", classification_report_train)

# Get the classification report for the validation data
print("[INFO] evaluation testing for validation data ...")
predictions_val = CNN3D_model.predict(features_val, batch_size=2)  # Change 'batch_size' to the desired value
classification_report_val = classification_report(labels_val.argmax(axis=1), predictions_val.argmax(axis=1), target_names=CLASSES_LIST)

# Print the classification report for validation data
print(classification_report_val)

# Get the classification report for the test data
print("[INFO] evaluation testing for test data ...")
predictions_test = CNN3D_model.predict(features_test, batch_size=2)  # Change 'batch_size' to the desired value
classification_report_test = classification_report(labels_test.argmax(axis=1), predictions_test.argmax(axis=1), target_names=CLASSES_LIST)

# Print the classification report for test data
print(classification_report_test)
```

Figure 3.14: Classification Report

```
# Evaluate your model on the test data and get the predicted labels
predicted_labels = CNN3D_model.predict(features_test)
predicted_labels = np.argmax(predicted_labels, axis=1)

true_labels = np.argmax(labels_test, axis=1)

# Compute the confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# Calculate the row sums (total samples per class)
row_sums = conf_matrix.sum(axis=1, keepdims=True)

# Normalize the confusion matrix to percentages
conf_matrix_percent = conf_matrix / row_sums

# Plot the confusion matrix with percentages
class_names = CLASSES_LIST  # Replace with your class names
fig, ax = plt.subplots(figsize=(10, 10))  # Adjust the figure size
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_percent, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues, values_format=".2f", ax=ax)
plt.xticks(rotation=45)
plt.title("Confusion Matrix (Percentages) 3DCNN")

for text in disp.text_.ravel():
    text.set_fontsize(12)
plt.show()
```

Figure 3.15: Confusion Matrix

3.3.3.6. Model Saving

After training the model and getting a satisfactory model performance, the model will be saved by using model.save to store the model in the Hierarchical Data Format (HDF5) format (.h5). This saved model can be loaded and used further in the application process.

```
# Get the loss and accuracy from model_evaluation_history.
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history
model_file_name='/content/drive/MyDrive/ProjectResearchMsc/Model/CNN3D_model_01.h5'
# Save the Model.
CNN3D_model.save(model_file_name)
```

Figure 3.16: Model Saving

3.3.3.7. Application Design

The overall process of the application design flow chart is shown in figure 3.17.

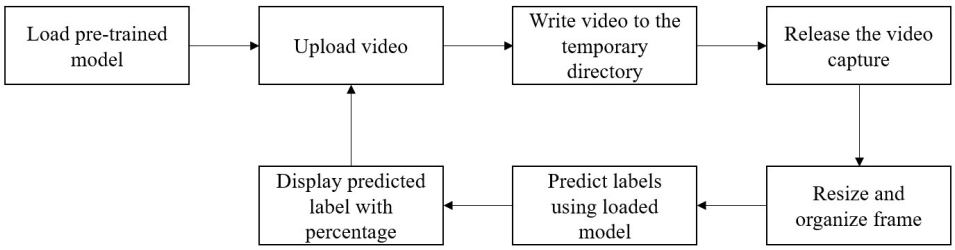

Figure 3.17: Application Flowchart

To implement the Sign Language Recognition application by using the Streamlit:

1. Model Integration: The pre-trained model will be loaded and compiled into the application using load_model function from TensorFlow/Keras.

```
# Load your pre-trained model and perform other operations
CNN3D_model = load_model('model/tested_model_3DCNN1.h5')
CNN3D_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'], run_eagerly=True)
CLASSES_LIST = ["book", "drink", "computer", "before", "chair", "go"]
```

Figure 3.18: Model Integration in Application

2. Video Processing: First, store the uploaded video in a temporary directory, then capture video frames and resize to 64x64 as the model's input shape. Normalise the frames to range [0 1] and organise frames into sequences (50 frames).

```python
# Use Streamlit's file_uploader to allow users to upload a video
video_path = st.file_uploader("Upload a video", type=["mp4", "MOV"])

if video_path is not None:
    st.write("Processing video...")
    st.video(video_path)

    # Create a temporary directory to store the uploaded video
    temp_dir = tempfile.mkdtemp()
    temp_video_path = os.path.join(temp_dir, "temp_video.mp4")

    # Write the uploaded video to the temporary directory
    with open(temp_video_path, "wb") as temp_video_file: ···
```

```python
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Resize frame to match model's input shape (64x64)
    frame = cv2.resize(frame, (64, 64))

    video_frames.append(frame)

cap.release()

# Normalize pixel values
video_frames = np.array(video_frames) / 255.0

# Organize frames into sequences with length 50
sequence_length = 50
num_frames = len(video_frames)
```

3. Model Prediction: Use the loaded model to predict class labels.

```python
predictions = CNN3D_model.predict(sequences)

# Assuming predictions contain class probabilities
predicted_class_indices = np.argmax(predictions, axis=-1)
predicted_labels = [CLASSES_LIST[i] for i in predicted_class_indices]

# Count the occurrences of each label
label_counts = Counter(predicted_labels)

# Calculate the percentage of each label
total_labels = len(predicted_labels)
label_percentages = {label: count / total_labels * 100 for label, count in label_counts.items()}

# Find the most frequent label
most_frequent_label = max(label_percentages, key=label_percentages.get)
```

Figure 3.19: Model Prediction in Application

4. Result Presentation: Present the most frequent predicted label as the output and display the original video.

```
st.write("Predicted Output:", most_frequent_label)
st.write("Predicted Labels and Percentages:")
for label, percentage in label_percentages.items():
    st.write(f"{label}: {percentage:.2f}%")

# Display the video using OpenCV's cv2_imshow
for frame in video_frames:
    cv2.imshow("Video", frame)
    cv2.waitKey(30)  # Adjust the delay (in milliseconds) between frames

cv2.destroyAllWindows()

# Cleanup temporary directory and video file
os.remove(temp_video_path)
os.rmdir(temp_dir)
```

Figure 3.20: Result Presentation in Application

Chapter 4

Software Development

Summary

This section presents an overview of the fundamental tools and machine environments chosen for the project's software development process. The selection of these tools relies on their suitability for the project's specific requirements and objectives, rather than personal familiarity.

4.1 Toolsets and Machine Environments

4.1.1 Development Tools Selection

In this section, an evaluation will be conducted on several development tools that are approachable for machine learning tasks. The selection process will involve considering their strengths and compatibility with the project's requirements.

4.1.1.1 Development IDE Tools

Local integrated development environments (IDEs)

Local IDE such as Visual Studio Code (VSCode) or Spyder provide robust functionalities for editing and debugging code in machine learning and data science projects. It is associated with libraries and facilitates developers to work in a familiar environment. Using local IDE allows users to install specific extensions, themes, and packages as preferences. It is offline access that does not depend on an internet connection and cloud resources. Moreover, users have full authority over their local development

environment, including selecting Python versions, libraries, and system configurations (Saabith, Vinothraj and Fareez, 2021).

However, it is essential to consider the following weaknesses of using a local IDE:

- Resource Limitations: Local IDEs rely on the developer's machine's hardware resources. This can result in computational and memory limitations, making it challenging to train deep learning models or process large datasets efficiently.
- Dependency Management: Python libraries and dependencies require manual installation and configuration, which is difficult to manage across team members.
- Collaboration Features: It is unsuitable for real-time collaboration, which is difficult when working in a team.

Google Colab

Google Colab is a cloud-based Jupyter Notebook that provides a wide range of libraries, tools, and resources for machine learning and data science approaches, which offers several advantages (colab.google, n.d.):

- Resource Provisioning: Google Colab offers users the opportunity to utilise high-performance GPUs and TPUs (Tensor Processing Units) to enhance the training process of deep learning models without any hardware changes on the machines.
- No Installation Required: Unlike local IDEs, Colab does not require users to install or manage libraries and dependencies directly. This simplified setup ensures the consistency of the software.
- Extensive Library Support: Google Colab is equipped with pre-installed machine learning frameworks such as TensorFlow as well as data visualisation tools such as Matplotlib.
- Collaboration: Google Colab offers real-time collaboration, which enables continuous participation among team members in the same notebooks. This facilitates the development of collaborative model creation and exchange of knowledge.

However, it is important to point out that Google Colab also have certain limitations due to a closed environment that prevents users from installing their own Python packages

or modules. Several times, the system suffers from unexpected disconnects or system crashes, resulting in loss of unsaved work if appropriate precautions are not taken. The free edition of Colab has limited resources, which may not be sufficient for training complicated models. In order to address this limitation, a subscription extension may be required. Lastly, it requires an internet connection, and if the connection is interrupted, the work may be paused (Pandiya, Dassani and Mangalraj, 2020).

In conclusion, Google Colab has been chosen as the development environment for this project based on its advantages, which are suitable for the project's requirements, especially in terms of machine learning model training, collaborative development, and data analysis. Despite its limitations, such as session duration, the cloud-based Google Colab provides access to powerful GPUs and TPUs, which eliminates the need for hardware upgrades.

4.1.1.2 Programming Languages

Python: Python is a popular programming language used for machine learning, data analysis, and managing data tasks. It is widely used for machine learning with widely used libraries and frameworks such as TensorFlow, PyTorch, and scikit-learn. It is readable and user-friendly, making it a great choice for data scientists and machine learning engineers. Moreover, it is also compatible with Google Colab Notebooks, which is selected as the development tool (Raschka, Patterson and Nolet, 2020).

JavaScript JavaScript is a famous programming language that is commonly used in web development to enhance the user experience. It can also be used for scientific computing, data science and machine learning. However, there are limitations and challenges to using JavaScript for machine-learning tasks due to it being a dynamically typed programming language, which lacks static type annotations, making it challenging to program and debug complex machine-learning algorithms that need precise data types and structures. This tool may not be as efficient as those for other languages like Python and might not be the best choice for conducting machine learning models (Andreasen et al., 2017).

Streamlit Streamlit is an open-source framework for Machine Learning to deploy the model into the web application. It was developed specifically for the Python program-

ming language, which simplifies web application developments in terms of integration with major Python libraries, including scikit-learn, Keras, PyTorch, NumPy, pandas, and Matplotlib, among others. Furthermore, employing machine learning models in web apps is more straightforward compared to manual HTML coding, for which the Library API is not necessary, such as Flask (LatentView Analytics, n.d.).

In conclusion, Python was selected for the machine learning approach due to its extensive library and framework support, simplifying the development and deployment of complex models, making it the preferred choice for this project. Within the context of web application development. Although JavaScript is widely used for enhancing frontend interactivity and user engagement, Streamlit was the preferred choice for web application development as it facilitates data-driven web applications with less coding required. This choice is particularly beneficial when time constraints limit the development to a prototype stage.

4.1.1.3 Development Environments and Version Control

For the development of this project, Google Colab has been selected as the primary development environment, as its superior built-in integration with GPU and TPU accelerates the training and experimentation, particularly in machine learning tasks for this project that require a large amount of computational resources to deal with sign language videos. Moreover, it provides free access to powerful computing resources and the option to upgrade to even more efficient hardware. Colab also has integration with version control systems such as Git, which can facilitate code management, team collaboration and allow users to monitor changes and updates easily. The Git repository's commit history allows for tracking of introduced features and bug fixes. Also, it is possible to make a connection between the Colab environment and the GitHub repository to ensure that the code remains organised, collaborative, and well-documented.

4.2 Implementation

This section will discuss and analyse software development challenges and solutions encountered during the implementation process.

4.2.1 Development Process

The project implementation involved several steps, including data preprocessing and model deployment. The steps are summarised below:

1. Data Preprocessing: The initial challenge was preparing the dataset for model training, which includes data grouping, frames extraction, data argumentation, one-hot encoding of labels and data splitting. It is crucial to handle the data with caution to ensure model robustness.

2. Model Development: The primary goal of this project is the development of machine learning models for the classification of sign language. The experiment of LRCN and 3DCNN has been conducted. A significant model development challenge was identifying the optimal architecture, hyperparameters, and model complexity regarding computational resources and accuracy.

3. Integration with Application: Integrating machine learning models with the Streamlit application to present the prototype of the model deployment into a web application. Correctness of prediction and real-time inference processing needs to be ensured to smooth user interactions.

4.2.2 Challenges and Solutions

During the implementation phase, several challenges were encountered:

1. Limited Dataset: One of the primary challenges was the limitation of the dataset, particularly the small number of samples in each sign language class. The limited availability of data presented a difficulty in terms of training the model and achieving high model performance. The sliding window method has been used as data argumentation to overcome this challenge to increase the training dataset. The employed method successfully increased the size of the dataset by creating more samples through the process of windowing existing data. This strategy improved the diversity of the training data, by boosting the model's training capabilities. The sliding window method was added after the frame extraction process and

used in creating the dataset for the training process, the stride is set to 5, as shown in figure 4.1.

```
# Specify the stride for the sliding window
STRIDE = 5
def sliding_window(sequence, window_size, stride):
    num_samples = (len(sequence) - window_size) // stride + 1
    sub_sequences = [sequence[i:i + window_size] for i in range(0, num_samples * stride, stride)]
    return np.array(sub_sequences)
```

Figure 4.1: Sliding Window Function

2. Hyperparameter Tuning and Overfitting: One additional problem occurred in the process of fine-tuning the model's hyperparameters. The goal is not only to utilise the selected parameters to enhance model performance but also to avoid the issue of overfitting. Models initially addressed signs of overfitting, where it performed well on the training data but struggled with unseen data. Dropout layers were added to the neural network architecture to overcome this challenge with a value of 0.5. Including dropout layers was valuable in mitigating overfitting by introducing a random deactivation mechanism for specific neurons throughout the training process. Furthermore, L2 regularisation was added to the models to improve their accuracy and make them more generalised with lambda equal to 0.001. These parameters can be adjusted to observe the model's reaction. The final model is shown in figure 4.2.

```
# Define the Model Architecture.
########################################################################################################

model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu'),
                          input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
model.add(TimeDistributed(MaxPooling2D((4, 4))))
model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((4, 4))))
model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))
model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same', activation='relu'), name='custom_conv2'))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

# Add L2 regularization to layers
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))

model.add(TimeDistributed(Flatten()))
model.add(LSTM(32)) #number of memory cells or units in the Long Short-Term Memory (LSTM) layer.
model.add(Dense(len(CLASSES_LIST), activation = 'softmax'))
```

Figure 4.2: Added Dropout and L2 layers

3. Colab Free Version Limitation: Another challenge encountered was the limitation of the free version of Google Colab. The free edition only offers limited access to computing resources, such as GPU access, which was not enough for training large-scale models. The paid version was used to enhance computational resources, significantly accelerating and stabilising the model training process.

4. Real-World Data Deployment: During the deployment process in web application using real-world data, which was obtained from many sources including mobile devices from various signers. It introduced variations, noise, and unexpected scenarios that were not fully represented in the trained data. This presents a challenge in maintaining the same level of accuracy achieved in the application. Since this project only performs a prototype of the application, the continuous improvement method will be utilised in future work by periodically retraining the models with updated datasets that reflect real-world scenarios.

4.3 Testing

In this section, the methodology employed for testing this project was implementing unit testing to evaluate the performance and functionality of individual components and tasks within the project.

4.3.1 Unit Testing

According to Agile and Extreme Programming methods that have been adopted in this project. In this section, unit testing was conducted on individual components of the model to ensure expected functionality and overall satisfaction of project requirements. The project was broken down into smaller iterations, which are data preprocessing, model training, and web application. For each module, unit test cases were created to verify correctness, functionality, and expected outcomes.

Data Processing: A Unit test case was created to evaluate the data processing iteration. These tests ensured that the data fed into the model was in the correct format. The data processing test case is shown in table 4.1.

No.	Test Case	Step	Description	Expected Result	Actual Result	Pass/Fail
1	Data Grouping	1. Read JSON File	Read the provided JSON file containing video IDs and labels.	JSON file is successfully read.	As Expected	Pass
		2. Group Data	Group the raw video data into correct category folders based on labels.	Data is organized into labeled folders.	As Expected	Pass
2	Frames Extraction	1. Extract Frames	Extract frames from video sequences.	Frames are successfully extracted.	As Expected	Pass
		2. Resize Frames	Resize frames to 64x64 dimensions.	Frames are resized to the specified dimensions.	As Expected	Pass
		3. Create Sequences	Create sequences of 50 frames each.	Sequences contain the correct number of frames.	As Expected	Pass
3	Data Augmentation	1. Perform Augmentation	Apply data augmentation techniques to increase dataset size.	Augmented data is generated.	As Expected	Pass
		2. Check Data Size	Verify the size of the augmented dataset.	Augmented dataset has a larger size.	As Expected	Pass
4	Create Dataset	1. Store Features and Labels	Store the augmented data as features and labels.	Dataset is successfully created.	As Expected	Pass
5	Split Data	1. Split Data	Split the dataset into training and testing sets (75:25 ratio).	Data is divided as specified.	As Expected	Pass

Table 4.1: Data Processing Test Case

Model Training: A unit test case was created to evaluate model training iterations, ensuring that the model is structured correctly. The model training test case is shown in table 4.2.

No.	Test Case	Step	Description	Expected Result	Actual Result	Pass/Fail
1	Check Model Architecture	1. Inspect Model Architecture	Inspect the architecture of the created model.	Model architecture is correctly defined.	As Expected	Pass
2	Model Created Successfully	1. Create the Model	Create the sign language classification model.	Model is created successfully.	As Expected	Pass
3	Compile Model	1. Compile the Model	Compile the model with specified settings.	Model is compiled without errors.	As Expected	Pass
4	Training the Model	1. Start Model Training	Begin training the model with training data.	Model trains without errors.	As Expected	Pass
5	Evaluate the Trained Model	1. Evaluate the Model	Evaluate the trained model using validation data.	Model evaluation metrics meet the criteria.	As Expected	Pass
		2. Graph Plot	Plot training and validation loss and accuracy.	Graphs of loss and accuracy are plotted correctly.	As Expected	Pass
6	Save Model in Correct Path	1. Save the Model	Save the trained model in the specified path.	Model is saved to the correct path without errors.	As Expected	Pass

Table 4.2: Model Training Test Case

Web Application: A unit test case was created to evaluate the web Application iterations, ensuring that the application matched the project requirements. The web application test case is shown in table 4.3.

No.	Test Case	Step	Description	Expected Result	Actual Result	Pass/Fail
1	Load Model	1. Load Trained Model	Load the trained sign language classification model.	Model is loaded without errors.	As Expected	Pass
2	Compile Model	1. Compile Model	Compile the model for use in the web application.	Model is compiled without errors.	As Expected	Pass
3	Load Input Video	1. Load Sample Video	Load a sample sign language video for testing.	Video is loaded successfully.	As Expected	Pass
4	Resize Input Video as Model Trained	1. Resize Video to Model Specifications	Resize the input video to match the model's requirements.	Video is resized to the correct dimensions.	As Expected	Pass
5	Interface Show Button to Upload Video	1. Display Upload Button	Present a button on the web interface to upload videos.	Upload button is displayed on the interface.	As Expected	Pass
6	Predicted Label Presented on Screen	1. Perform Prediction	Perform sign language gesture prediction on uploaded video.	Predicted label is displayed on the screen.	As Expected	Pass

Table 4.3: Web Application Test Case

4.3.2 Model Performance Evaluation

In this section, the performance of a fully trained machine learning model has been evaluated for sign language classification.

Model Accuracy Metrics: In order to assess the model performance, a variety of accuracy metrics have been measured, including accuracy, precision, recall, F1-score, and support, which offer valuable insight into the efficiency of the model. This project will mainly focus on the f1-score to specify the performance in each class prediction because it represents the harmonic mean of both precision and recall. Likewise, the weighted average will be used to specify the model performance in each training, validation and testing process across multiple classes.

Confusion Matrix Visualization: The confusion matrix is a powerful tool in terms of providing a visual representation of the outcomes of a categorisation model. By displaying the true positives, true negatives, false positives, and false negatives, it allows a deeper understanding of the accuracy and effectiveness of the model.

Model Evaluation Metrics Calculation

To evaluate the machine learning model performance, precision, recall, and f1-score have been used as described below:

Accuracy: The accuracy can be calculated from the proportion of correctness predictions on all predictions made, as shown in equation 4.1.

$$Accuracy = \frac{(TP + TN)}{TP + FP + TN + FN} \tag{4.1}$$

Precision: The precision can be calculated by the number of true positives divided by total positives, as shown in equation 4.2.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall: The recall can be calculated by the number of true positives divided by the total of true positives and false negatives, as shown in equation 4.3.

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

F1 Score: The f1 score is a famous metric for evaluating the performance of a classification model; it can calculated by harmonic means of precision and recall, as shown in equation 4.4.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

Where TP is true positive, TN is true negative, FP is false positive, and FN is false negative (Koo P.S, 2018).

Moreover, various averaging methods for F1 score calculation have been used in multi-class classification, including macro, weighted, and micro averages (Leung, K, 2022).

Macro Average: It is a straightforward method that can be calculated by the arithmetic mean or unweighted mean of all classes in terms of f1 score. Using this method is inefficient as it treats all classes equally without considering support values, as shown in equation 4.5.

$$MacroAvg(f1) = \frac{1}{n}\sum_{i=1}^{n} f1Score_i \tag{4.5}$$

Micro Average: It represents the proportion of correctly classified overall factors which can be calculated by the global average of the f1 score by summing the TP, FP and FN across all classes and then putting it into the f1 score equation, as shown in equation 4.6.

$$MicroAvg(f1) = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.6}$$

Weighted Average: It is calculated from the mean of all classes in the f1 score while considering supports from each class. Where support indicated the number of actual occurrences of each class. It is better to use this metric when dealing with an unbalanced dataset. The equation for weighted average is shown in equation 4.7. Where w is support and X_i is Per-class f1 score.

$$WeightedAvg(f1) = \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i} \tag{4.7}$$

Confusion Metric: A confusion matrix is a summary of the prediction in matrix form. It displays the number of correct and incorrect predictions per class, which helps clarify the classes the model is mistaking for other classes. Where TP stands for True Positive Predictions. TP will be equal to 1 if it is correctly labelled as 1 or the number of fraudulent transactions that were labelled as fraudulent. TN stands for True Negative Predictions, which means the amount of 0, or transactions that are not fraudulent and are labelled as 0. FP (False Positive) is the number of non-fraudulent transactions that were marked as fraud, and FN (False Negative) is the number of fraud transactions that were marked as non-fraudulent (Tiwari, 2022).

Chapter 5

Results

Summary

In this section, the results of the model performance evaluation will be presented, focusing on two key models: LRCN and 3DCNN. The effect of integrating the sliding window method, as well as the addition of dropout layers and L2 regularisation. The objective is to provide a clear and organised summary of the model performance throughout the development process. The discussion of the result will be presented in *section 6: Discussion and Evaluation.*

5.1 LRCN Model Performance Results

5.1.1 Performance Before Sliding Window

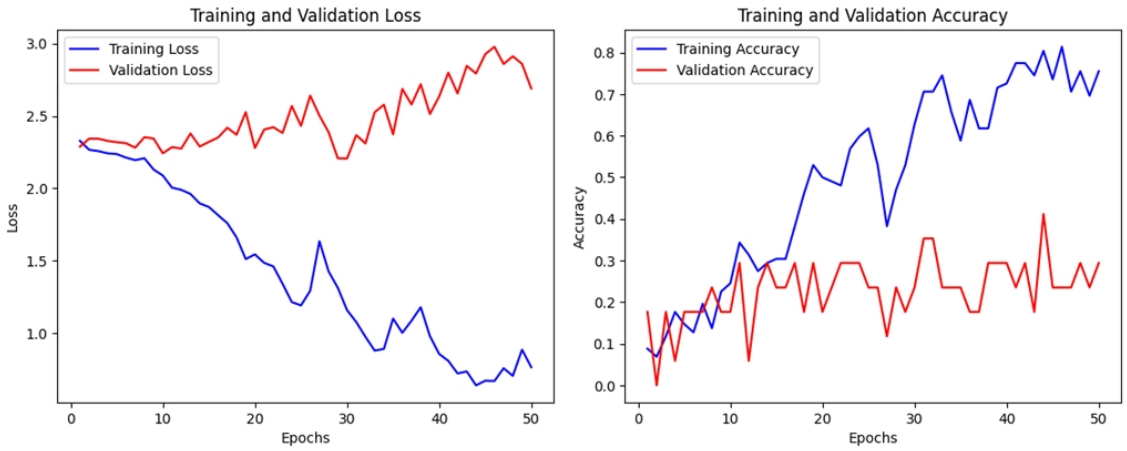

Figure 5.1: Training Loss and Accuracy Plot Before Applying Sliding Window

5.1.2 Performance After Sliding Window

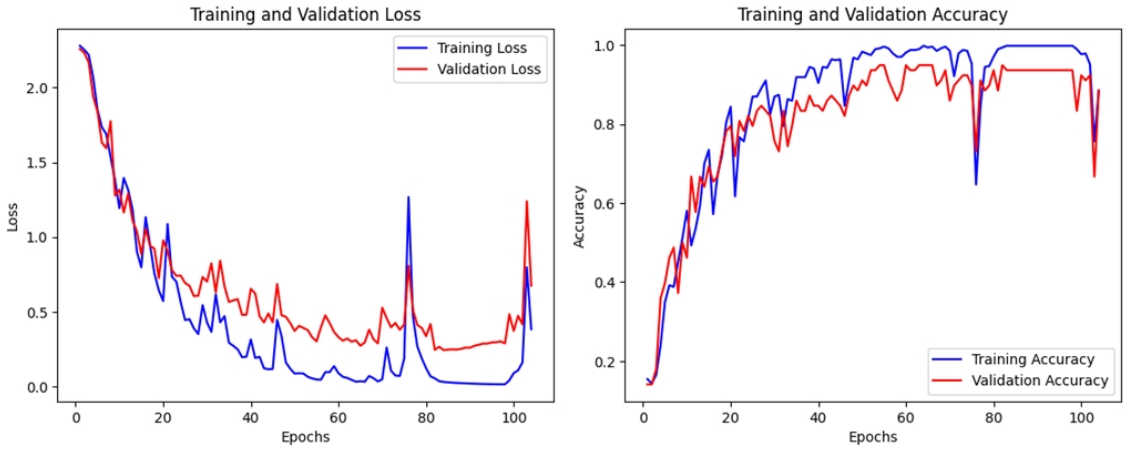

Figure 5.2: Training Loss and Accuracy Plot After Applying Sliding Window

5.1.3 Performance After Adding Dropout and L2 Regularisation

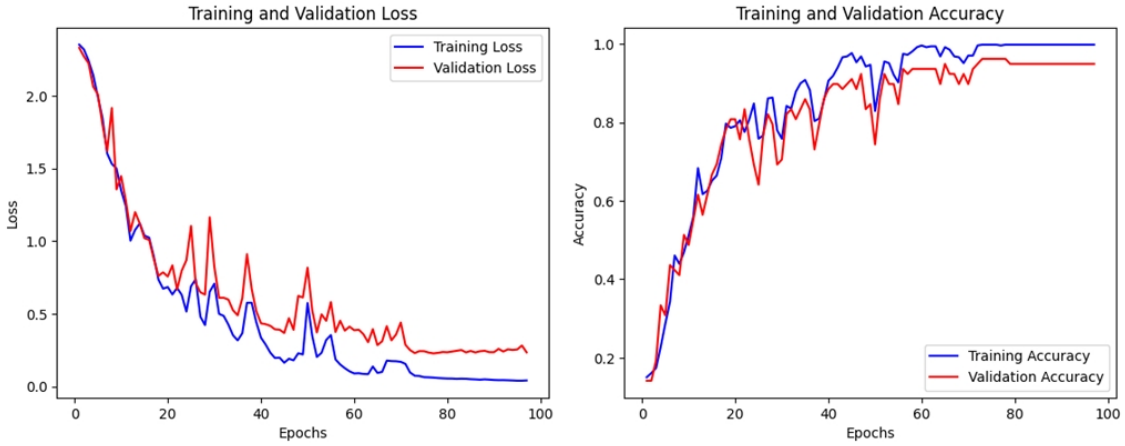

Figure 5.3: Training Loss and Accuracy Plot After Applying Dropout and L2

Confusion Matrix Visualization:

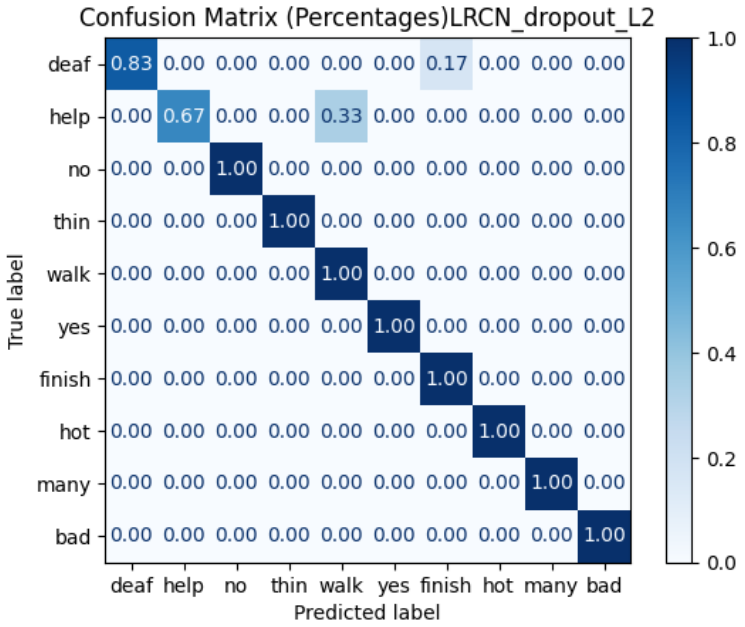

Figure 5.4: Confusion Matrix Plot After Applying Dropout and L2

Model Accuracy Metrics:

Class	precision	recall	f1-score	support
deaf	1.00	0.98	0.99	63
help	1.00	1.00	1.00	36
no	0.98	1.00	0.99	60
thin	1.00	1.00	1.00	42
walk	1.00	1.00	1.00	77
yes	1.00	1.00	1.00	41
finish	1.00	1.00	1.00	47
hot	1.00	1.00	1.00	46
many	1.00	1.00	1.00	58
bad	1.00	1.00	1.00	28
accuracy			1.00	498
macro avg	1.00	1.00	1.00	498
weighted avg	1.00	1.00	1.00	498

Table 5.1: Model Accuracy Metrics: Training Data

Class	precision	recall	f1-score	support
deaf	1.00	0.92	0.96	13
help	1.00	1.00	1.00	4
no	0.80	1.00	0.89	8
thin	1.00	1.00	1.00	7
walk	1.00	1.00	1.00	5
yes	1.00	1.00	1.00	6
finish	1.00	1.00	1.00	3
hot	1.00	0.67	0.80	6
many	1.00	1.00	1.00	7
bad	0.75	1.00	0.86	3
accuracy			0.95	62
macro avg	0.96	0.96	0.95	62
weighted avg	0.96	0.95	0.95	62

Table 5.2: Model Accuracy Metrics: Validation Data

Class	precision	recall	f1-score	support
deaf	0.83	0.83	0.83	6
help	1.00	0.67	0.80	3
no	0.80	1.00	0.89	4
thin	1.00	0.90	0.95	10
walk	1.00	1.00	1.00	13
yes	1.00	1.00	1.00	4
finish	0.80	1.00	0.89	4
hot	1.00	1.00	1.00	5
many	1.00	1.00	1.00	10
bad	1.00	1.00	1.00	4
accuracy			0.95	63
macro avg	0.94	0.94	0.94	63
weighted avg	0.96	0.95	0.95	63

Table 5.3: Model Accuracy Metrics: Testing Data

Execution Time: 252.71 seconds

Resource Used:

```
!nvidia-smi

Wed Aug 30 23:35:33 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   39C    P0    39W / 300W |   9398MiB / 16384MiB |      2%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Figure 5.5: Resource Used in LRCN

5.2 3DCNN Model Performance Results

5.2.1 Performance Before Sliding Window

Figure 5.6: Training Loss and Accuracy Plot Before Applying Sliding Window

5.2.2 Performance After Sliding Window

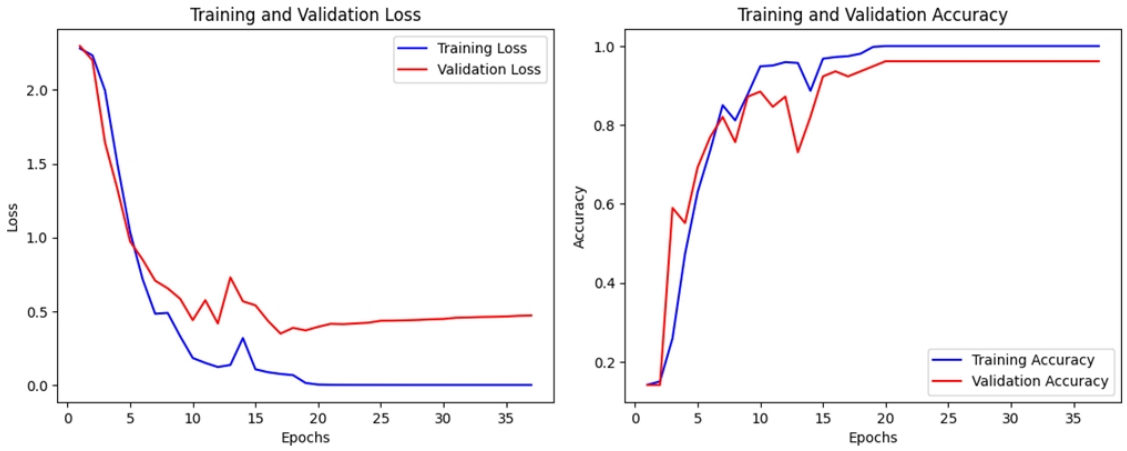

Figure 5.7: Training Loss and Accuracy Plot After Applying Sliding Window

5.2.3 Performance After Adding Dropout and L2 Regularisation

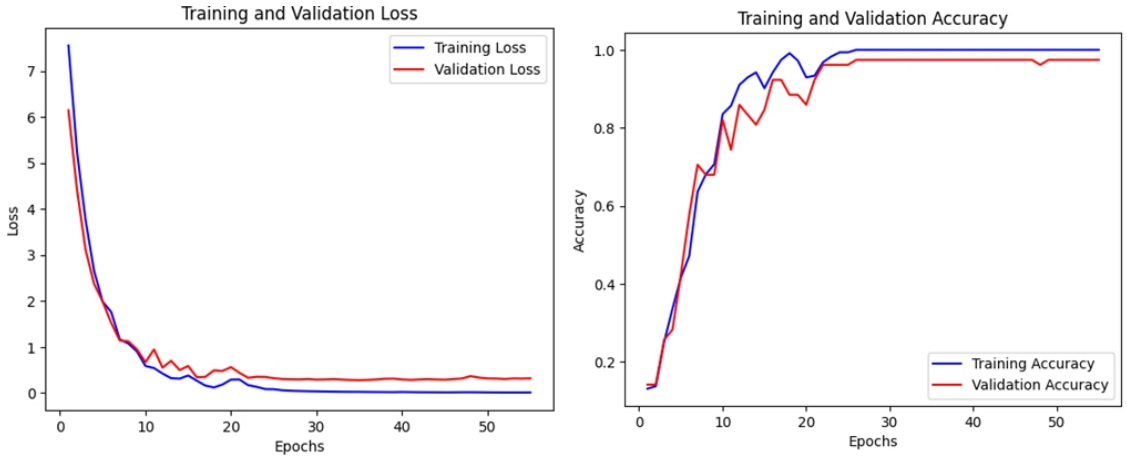

Figure 5.8: Training Loss and Accuracy After Applying Dropout and L2

Confusion Matrix Visualization:

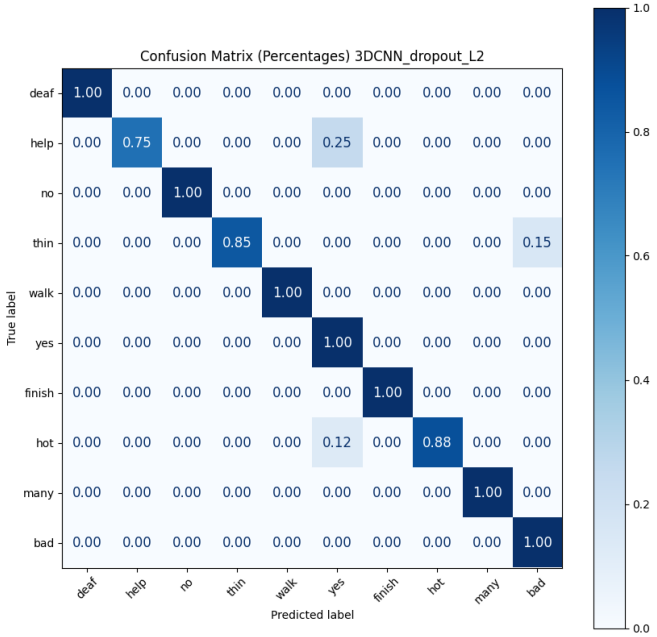

Figure 5.9: Confusion Matrix Plot After Applying Dropout and L2

Model Accuracy Metrics:

Class	precision	recall	f1-score	support
deaf	1.00	1.00	1.00	59
help	1.00	1.00	1.00	32
no	1.00	1.00	1.00	59
thin	1.00	1.00	1.00	36
walk	1.00	1.00	1.00	74
yes	1.00	1.00	1.00	39
finish	1.00	1.00	1.00	44
hot	1.00	1.00	1.00	45
many	1.00	1.00	1.00	51
bad	1.00	1.00	1.00	28
accuracy			1.00	467
macro avg	1.00	1.00	1.00	467
weighted avg	1.00	1.00	1.00	467

Table 5.4: Model Accuracy Metrics: Training Data

Class	precision	recall	f1-score	support
deaf	1.00	0.85	0.92	13
help	1.00	1.00	1.00	7
no	1.00	1.00	1.00	9
thin	1.00	1.00	1.00	10
walk	1.00	1.00	1.00	11
yes	0.86	1.00	0.92	6
finish	1.00	0.80	0.89	5
hot	1.00	0.75	0.86	4
many	0.91	1.00	0.95	10
bad	0.60	1.00	0.75	3
accuracy			0.95	78
macro avg	0.94	0.94	0.93	78
weighted avg	0.96	0.95	0.95	78

Table 5.5: Model Accuracy Metrics: Validation Data

Class	precision	recall	f1-score	support
deaf	1.00	1.00	1.00	10
help	1.00	0.75	0.86	4
no	1.00	1.00	1.00	4
thin	0.92	0.85	0.88	13
walk	1.00	1.00	1.00	10
yes	0.75	1.00	0.86	6
finish	1.00	1.00	1.00	5
hot	1.00	0.88	0.93	8
many	0.93	0.93	0.93	14
bad	0.80	1.00	0.89	4
accuracy			0.94	78
macro avg	0.94	0.94	0.93	78
weighted avg	0.94	0.94	0.94	78

Table 5.6: Model Accuracy Metrics: Testing Data

Execution Time: 219.69 seconds

Resource Used:

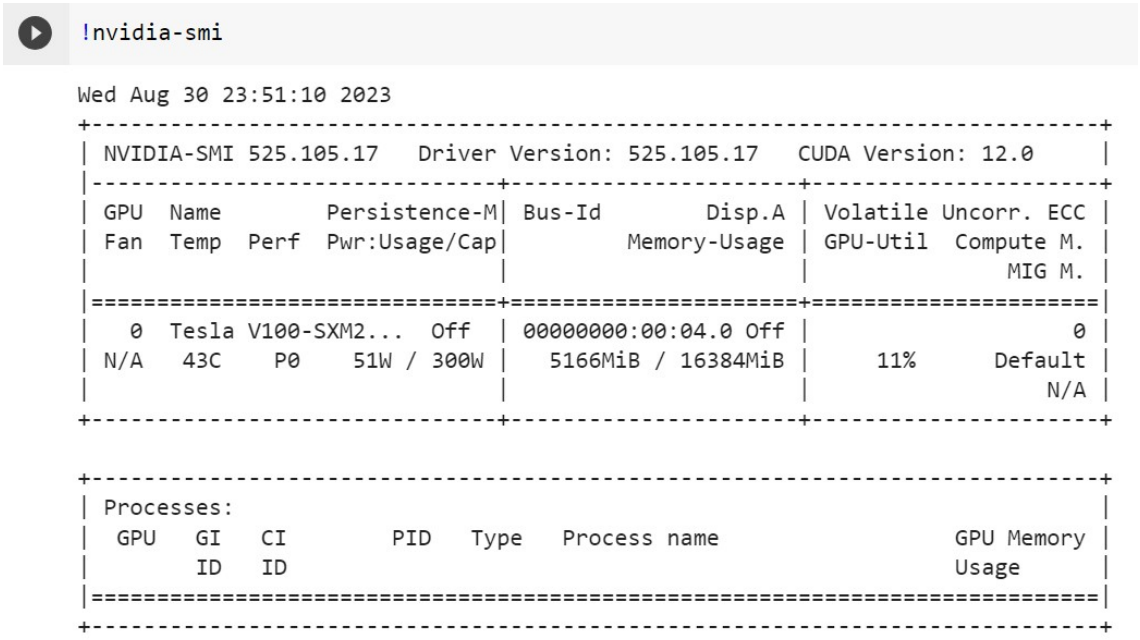

Figure 5.10: Resource Used in 3DCNN

5.3 Summary Result for LRCN and 3DCNN

The summarised model performance for LRCN and 3DCNN are presented in table 5.7 and 5.8 in which f1-score specifies the performance in each class prediction and the weighted average specifies the model performance across multiple classes.

Category	F1-score		
	Train	Validation	Test
deaf	0.99	0.96	0.83
help	1	1	0.8
no	0.99	0.89	0.89
thin	1	1	0.95
walk	1	1	1
yes	1	1	1
finish	1	1	0.89
hot	1	0.8	1
many	1	1	1
bad	1	0.86	1
weighted avg	1	0.95	0.95

Table 5.7: Summary Result LRCN Model

Category	F1-score		
	Train	Validation	Test
deaf	1	0.92	1
help	1	1	0.86
no	1	1	1
thin	1	1	0.88
walk	1	1	1
yes	1	0.92	0.86
finish	1	0.89	1
hot	1	0.86	0.93
many	1	0.95	0.93
bad	1	0.75	0.89
weighted avg	1	0.95	0.94

Table 5.8: Summary Result 3DCNN Model

5.4 Web Application Result

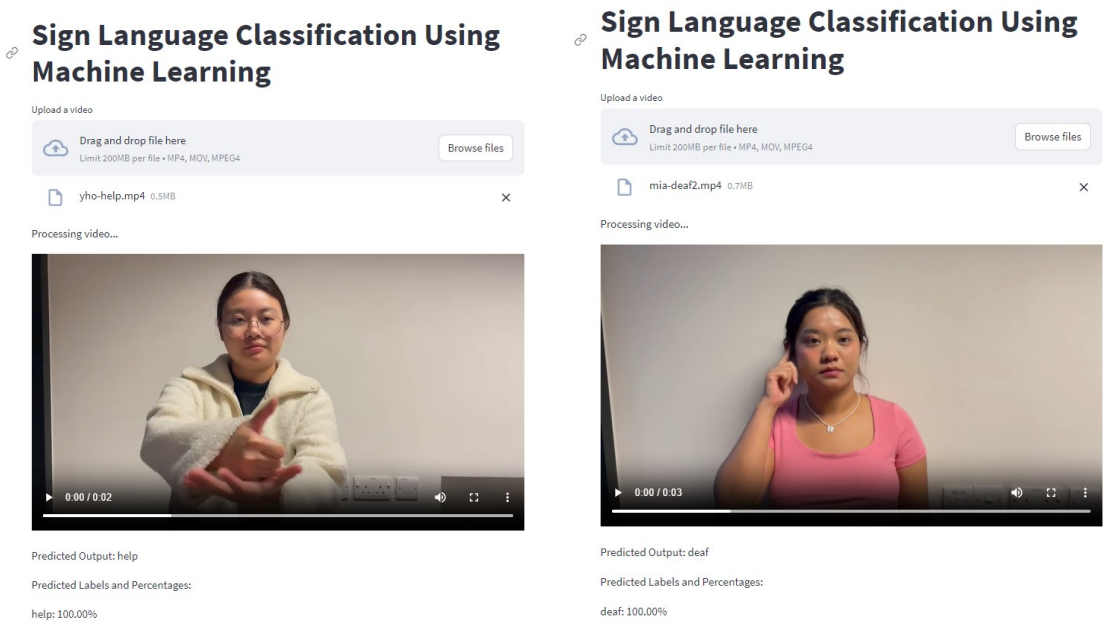

Figure 5.11: Web Application Interface

Chapter 6

Discussion and Evaluation

Summary

This section will critically discuss and evaluate the software solution in the context of the project's aims and objectives to provide an analysis of the results presented in *section 5: Result* and to reflect on the effectiveness and efficiency of the solution.

6.1 Model Development Analysis

This section will analyse the findings obtained from the baseline approach, followed by the integration of the sliding window method and dropout with L2 regularisation, in order to determine if the observed outcomes align with the predicted expectations.

6.1.1 Before Sliding Window

The baseline methods proposed for sign language gesture prediction for both LRCN and 3DCNN have been implemented. However, during the development process, two significant challenges were faced, which were overfitting and low accuracy in the training phase. Overfitting, as shown in figure 5.1 and 5.6, the validation loss was increased while the training loss decreased. It can be indicated as a significant sign of overfitting, where the model was fitting the training data too perfectly and struggling to perform in unseen data. Simultaneously, low validation accuracy during training was observed. This issue highlights the limited ability of the model to generalise beyond the trained data. As expected, the limitation of the available dataset would prevent the model's ability to

learn effectively. One of the best solutions to improve the model performance is data argumentation, in which a sliding window method was applied.

6.1.2 After Perform Sliding Window

After implementing the Sliding Window method to enhance the dataset to increase the model's ability to capture temporal patterns in sign language gestures, some improvements have been observed. As shown in figure 5.2 and 5.7, both training and validation loss were decreased while accuracy increased. However, the challenges remained, particularly regarding the fluctuations in the training loss and accuracy plots that might affect the overall accuracy. To improve the model's stability and robustness, L2 regularisation and dropout layer were added to the previous model.

6.1.3 After Adding Dropout and L2 Regularisation

Expanding on the Sliding Window approach, the next step in improving the model is to add a dropout and L2 regularisation layer into the model. These techniques were employed to address the fluctuations in training loss and accuracy plots, further enhancing the model's stability and generalisation performance. After integrating dropout layers and L2 regularisation, improvements were observed in the model's performance. One of the significant improvements was the reduction in fluctuation during the training process. These regularisation techniques improve the stabilising of the training loss and accuracy plots. So, this model has been selected as the best model for the LRCN and 3DCNN approach.

6.2 Model Comparative Analysis

This section will focus on a comprehensive comparative analysis between two significant models for sign language gesture recognition, which are LRCN and 3DCNN. The LRCN model is a sequential methodology that integrates the capabilities of Convolutional Neural Networks (CNNs) for extracting spatial features and Long Short-Term Memory (LSTM) networks for modelling temporal aspects. On the other hand, the 3DCNN model operates in the spatiotemporal domain directly, which makes use of the

3D convolutional layers to capture both spatial and temporal information simultaneously. In this comparative analysis, a detailed examination will be presented regarding the performance of these models in different scenarios, including their performance before and after the integration of the Sliding Window method, dropout layers and L2 regularisation. Valuable insights into choosing a suitable model architecture for sign language gesture recognition tasks will be provided by comparing their strengths and weaknesses.

6.2.1 Overall Model Performance

Upon evaluating the overall performance of both models from table 5.7 summary of the LRCN model, it achieved consistently high F1-scores across each class, and the weighted average was 1.0, 0.95, and 0.95 for training, validation, and test datasets, respectively. This indicates the LRCN model had an outstanding performance in sign language classification, demonstrating its effectiveness in learning and generalising from the dataset. Whereas, the 3DCNN model also revealed impressive performance. The weighted average for 3DCNN was 1.0, 0.95, and 0.94 for training, validation, and test datasets, which the test data score was slightly lower than the LRCN model.

6.2.2 Class-Specific Performance

Upon analysis of the F1-scores to see insights into the performance in each class, both models achieved high F1-scores in the testing dataset for most sign language gesture categories (above 0.8). From the result, some gestures have higher scores in the LRCN model but lower in 3DCNN. For "deaf" has 0.83 in LRCN but 1 in 3DCNN, which indicates that the LRCN model performed better in classifying this gesture than 3DCNN. However, there are other gestures that 3DCNN performs better than LRCN. So, the model performance alone may not be sufficient to determine which model is superior for this task. Therefore, the resources utilised will also be evaluated in the next section.

6.2.3 Resource Usage

Both models have been trained by using the same GPU, which is Tesla V100-SXM2 on Google Colab. As shown in figure 5.5 and 5.10, resource used result, the GPU utilised 9398 MiB out of a total capacity of 16384 MiB for training the LRCN model, while training the 3DCNN model used 5166 MiB. The higher GPU memory usage of the LRCN model compared to the 3DCNN model is likely because the LRCN model required more training epochs to achieve high accuracy compared to the 3DCNN model. Moreover, during the model training, execution time was measured. The LRCN took 252.71 seconds, while the 3DCNN took 219.69 seconds, which is slightly different. From the resource usage, it can be indicated that the most supportive evidence to compare these two models is memory usage, in which 3DCNN requires less memory than LRCN.

6.3 Web Application Analysis

This section will show the analysis of the web application that was developed for recognising sign language gestures. The evaluation covers various aspects, including functionality, and practicality.

6.3.1 Integration with Machine Learning Model

The machine learning model (.h5) can be loaded, compiled and integrated into the application correctly. Moreover, the web application is designed to be able to handle enhanced models, which can involve integrating larger datasets or alternative machine learning model approaches. The application will load the pre-trained models and use them to predict sign language gestures accurately. This flexibility ensures that the application is adaptable and scalable, making it easy to adapt to different datasets or model architectures.

6.3.2 Functionality and Sign Recognition

The core functionality of the web application is its ability to recognise and interpret sign language gestures accurately. The web application enables users to upload videos from their local storage. Upon upload, the interface displays the original video along with the predicted sign language gesture labels, complete with the corresponding percentage of confidence in the prediction at the bottom of the page. Moreover, the application's efficiency allows for quick video re-uploads and further interactions, ensuring a user-friendly and responsive environment with short time processing.

6.3.3 Real-World Applicability

In real-world scenarios, the application was tested with videos from real users to evaluate its robustness and practicality. It is essential to be aware that when transitioning from controlled datasets to real-world data, there can be challenges that may impact the accuracy of predictions. These challenges may be caused by variations in lighting, background, hand positioning, noises and other factors not present in controlled environments. As a result, some sign language gestures may not be correctly predicted due to the model's exposure to real-world complexities. This challenge leads to further research questions on improving this project to maintain prediction accuracy in diverse real-world environments.

6.4 Comparison with Project Objectives

In this section, the success of the project requirements and its objectives will be assessed. The objectives, as outlined in *section 1.2: Aim and Objectives*, will used as a guideline for evaluating the progress and achievements made.

Objective1: Develop a Robust Sign Language Recognition System

LRCN and 3DCNN model has been designed and developed to perform a sign language gesture recognition system. Both models were successfully created and implemented with high performance in interpreting and translating sign language gestures into text.

Objective2: Enhance Model Performance

The models have been trained and optimised by performing the sliding window method and adding dropout and L2-regularisation layers to enhance model performance. The results demonstrate the improvement in accuracy and robustness compared to the proposed baseline models.

Objective3: Evaluate the Model Performance

The evaluation metrics for both methods have been calculated and plotted to show insights into model performance and to analyse the strengths and weaknesses of the models. However, since the performances of the models are very close, it is important to measure resource usage as well, as the evaluation metrics alone may not be sufficient to select the best model.

Objective4: Analyse the Strengths and Limitations

A comparative analysis of two machine learning models that have been used in the project was conducted. This evaluation assesses the advantages, limitations, and performance of each model, providing valuable insights and fulfilling this requirement objective.

Objective5: Deploy machine learning model into web application

The pre-trained model has been loaded into the application which makes the project has successfully created a web application that allows users to upload sign-language gesture videos to see the prediction of the label.

Objective6: Evaluate the real-world applicability

The system's real-world applicability was addressed by extending its capabilities to handle variations in real-world data, which involved adapting it to different scenarios and situations. During this process, several challenges were encountered, such as wrong gesture prediction, which required several modifications to the system. However, despite these challenges, the system demonstrates a great prototype in the sign language recognition web application.

6.5 Reflective Analysis

6.5.1 Amendments and Changes

1. **Model Selection:** A single machine learning model was initially planned for developing a sign language recognition application. However, as the project progressed, it was decided to incorporate two distinct models, LRCN and 3DCNN. This change allowed for a more in-depth exploration of these models' comparative advantages and disadvantages.

2. **Enhancements:** During the initial stages of the project, the model architecture was designed without incorporating essential techniques such as data augmentation (sliding window), dropout layers, and L2 regularisation. At first, the project relied only on hyperparameter tuning to achieve optimal model performance. However, as the project progressed, it became evident that this approach was insufficient as the model failed to meet the desired standards.

3. **Resource Evaluation:** Although the initial approach did not particularly address resource efficiency, it became a significant component during the development of the model. Specifically, GPU memory usage became a priority to ensure that the system operates efficiently, especially on hardware with limited resources.

4. **Real-World Applicability:** The real-world applicability was initially aimed for this project. However, the complexities associated with handling variations in real-world data, such as lighting conditions, background, noise, and diverse signer styles, were underestimated.

6.5.2 What Went Well

To overcome the limitation of training data in terms of increased data size, the sliding window method was employed. However, there are several other techniques available to address this issue. The sliding window performed very well on both LRCN and 3DCNN models, which significantly boosted the training accuracy compared to the initial model proposal. After that, the dropout and L2 regularisation that was added into both models can reduce the fluctuations of training loss and accuracy effectively. Additionally, the structures of data processing and model structures used in both models are similar, resulting in reduced complexity and time required to perform and compare both models. Finally, the web application was developed by using Streamlit, which simplifies the loading of the pre-trained machine learning model and makes predictions. Moreover, updating the model is easy if we need to scale up the dataset.

6.5.3 What Didn't Go Well

The American Sign Language (ASL) dataset has a limitation of not being widely published online, even though ASL is one of the most commonly used sign languages in the world. It's not easy to find accurate videos with labels, and this lack of reliability of training resources results in low model robustness and practicality used in real-world environments. Therefore, data argumentation needs to be performed in this project. On training, validation and testing datasets, both models achieved high accuracy. However, when used with real signers in real environments, accuracy decreased, leading to incorrect classification of sign language gestures. To address this challenge, further research on noise, background removal or collecting more data needs to be conducted.

6.6 Future Directions

There are a number of exciting possibilities to pursue in future work to improve sign language recognition systems. A critical component is enhancing the robustness of these systems when applied to real-world datasets. This can be done by collecting more data from a real environment and using them to update during the training process. Updat-

ing the model with this real-world data will enhance the model to learn from various aspects. Moreover, before using the data to train the model, processing it for noise removal is essential. This ensures that the model learns only the relevant data, preventing it from being fooled by unnecessary noise. Investigating other methods, such as vision transformers capable of image classification and labelling, may also be worthwhile. Another direction to consider is to use a skeleton-based approach, which focuses on key movements instead of entire video frames. This can result in more accurate and efficient recognition. Finally, one important area of development is the creation of sign language recognition applications that can provide real-time communication support. These applications have the potential to bridge communication gaps and promote equality for people with hearing impairments.

Chapter 7

Conclusions

Summary

Sign language is an essential tool for communication in the deaf community. However, it remains a barrier for those who are not familiar with it. This project's aim is to develop a sign language recognition system to facilitate communication for people with hearing impairments by integrating a machine learning model into a prototype web application to reduce communication gaps between sign and non-sign users. The two methods that have been used to develop the machine learning models are a Long-term Recurrent Convolutional Network (LRCN), which is the combination of a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) network, and a 3D convolutional neural network. In which CNN is used to perform feature extraction and LSTM for sequence processing. Whereas, 3DCNN is capable of handling the requirements for video recognition. These two models were performed on the Word-Level American Sign Language (WLASL) video dataset. Due to the limitation of resources, the demonstrated dataset was selected as a prototype of this project in 10 classes. Challenges such as a limited dataset, poor training performance, and overfitting were faced during the development process. The solutions to migrate these challenges were employing Sliding Window as data argumentation and adding dropout and L2 regularisation layers. As a result, combining these techniques gave the high model performance on testing data in which the f1-score was more than 80% of each class and the weighted average of 95% and 94% in LRCN and 3DCNN, respectively. As the model performance in terms of prediction is similar, the resources used have been evaluated. It turned out that, LRCN

required more memory and time as more epochs were needed to get as high accuracy as 3DCNN. Furthermore, In the application that deployed the model with a real-world dataset, the accuracy was dropped due to the variations of uncontrolled environments, which can be improved in future work.

7.1 Future Work

Moving forward, several improvements in research could contribute to further robustness and practicality used in real-world data for sign language recognition systems. Firstly, noise removal from the input dataset should be performed to reduce the unnecessary information that may fool the model. Moreover, the project should consider expanding more datasets used for training the model to learn a wide range of pattern diversity of each sign gesture. One of the interesting techniques is continuous learning, which uses real-world data or self-collected data and feedback to the model for training. Further exploration of alternative machine learning approaches, such as the vision transformer model, is worth implementing and comparing with the proposed models, which also have the ability to do various natural language processing tasks and should be investigated in the context of sign language. Furthermore, using skeletal tracking data with video inputs may enhance the model's capacity to capture essential hand and body movements. Finally, the development of real-time sign language recognition applications should also be considered, as it can be practically used in general. In conclusion, this project aims to create a sign language recognition application using machine learning models. Additionally, future research attempts will be proposed to develop more robust, accurate, and accessible solutions.

References

Adenowo, Adetokunbo AA and Basirat A Adenowo (2013). 'Software engineering methodologies: a review of the waterfall model and object-oriented approach'. In: *International Journal of Scientific & Engineering Research* 4.7, pp. 427–434 (cit. on p. 13).

Albawi, Saad, Tareq Abed Mohammed and Saad Al-Zawi (2017). 'Understanding of a convolutional neural network'. In: *2017 international conference on engineering and technology (ICET)*. Ieee, pp. 1–6 (cit. on p. 7).

Alfonso, Maria Isabel and Antonio Botia (2005). 'An iterative and agile process model for teaching software engineering'. In: *18th Conference on Software Engineering Education & Training (CSEET'05)*. IEEE, pp. 9–16 (cit. on p. 13).

Alshamrani, Adel and Abdullah Bahattab (2015). 'A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model'. In: *International Journal of Computer Science Issues (IJCSI)* 12.1, p. 106 (cit. on p. 13).

Andreasen, Esben et al. (2017). 'A survey of dynamic analysis and test generation for JavaScript'. In: *ACM Computing Surveys (CSUR)* 50.5, pp. 1–36 (cit. on p. 34).

Aparna, C and M Geetha (2020). 'CNN and stacked LSTM model for Indian sign language recognition'. In: *Machine Learning and Metaheuristics Algorithms, and Applications: First Symposium, SoMMA 2019, Trivandrum, India, December 18–21, 2019, Revised Selected Papers 1*. Springer, pp. 126–134 (cit. on p. 6).

Basnin, Nanziba, Lutfun Nahar and Mohammad Shahadat Hossain (2020). 'An integrated CNN-LSTM model for Bangla lexical sign language recognition'. In: *Proceedings of International Conference on Trends in Computational and Cognitive Engineering: Proceedings of TCCE 2020*. Springer, pp. 695–707 (cit. on p. 6).

Bir, Paarth and Valentina E Balas (2020). 'A review on medical image analysis with convolutional neural networks'. In: *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE, pp. 870–876 (cit. on p. 11).

Camgoz, Necati Cihan et al. (2020). 'Sign language transformers: Joint end-to-end sign language recognition and translation'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10023–10033 (cit. on p. 6).

Carneiro, Tiago et al. (2018). 'Performance analysis of google colaboratory as a tool for accelerating deep learning applications'. In: *IEEE Access* 6, pp. 61677–61685 (cit. on p. 17).

Carroll, John and David Morris (2015). *Agile project management in easy steps*. In Easy Steps (cit. on p. 17).

Cheok, Ming Jin, Zaid Omar and Mohamed Hisham Jaward (2019). 'A review of hand gesture and sign language recognition techniques'. In: *International Journal of Machine Learning and Cybernetics* 10, pp. 131–153 (cit. on p. 2).

colab.google (n.d.). *Google Colaboratory*. URL: https://colab.google/. (cit. on p. 33).

Donahue, Jeffrey et al. (2015). 'Long-term recurrent convolutional networks for visual recognition and description'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634 (cit. on p. 9).

Huang, Jie et al. (2018). 'Attention-based 3D-CNNs for large-vocabulary sign language recognition'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.9, pp. 2822–2832 (cit. on p. 6).

Keras (2019). *Keras*. URL: https://keras.io/ (cit. on p. 20).

Koo P.S (2018). *Accuracy, Precision, Recall or F1?* URL: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. (cit. on p. 41).

Kusters, Annelies (2021). 'International Sign and American Sign Language as different types of global deaf lingua francas'. In: *Sign Language Studies* 21.4, pp. 391–426 (cit. on p. 2).

Lashgari, Elnaz, Dehua Liang and Uri Maoz (2020). 'Data augmentation for deep-learning-based electroencephalography'. In: *Journal of Neuroscience Methods* 346, p. 108885 (cit. on p. 10).

LatentView Analytics (n.d.). *Introduction to Streamlit*. URL: https://www.latentview.com/data-engineering-lp/introduction-to-streamlit (cit. on p. 35).

Leung, K (2022). *Micro, Macro Weighted Averages of F1 Score, Clearly Explained*. URL: https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f (cit. on p. 41).

Li, Dongxu et al. (2020). 'Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison'. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 1459–1469 (cit. on pp. 2, 19).

Malik, Rabia Saeed, Sayed Sayeed Ahmad and Muhammad Tuaha Hammad Hussain (2019). 'A review of agile methodology in IT projects'. In: *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)* (cit. on p. 13).

Matplotlib (2012). *Matplotlib: Visualization with Python*. URL: https://matplotlib.org/ (cit. on p. 20).

Narang, Rishi and Upendra Pratap Singh (2023). 'Interpretable Sequence Models for the Sales Forecasting Task: A Review'. In: *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, pp. 858–864 (cit. on p. 9).

Nimisha, KP and Agnes Jacob (2020). 'A brief review of the recent trends in sign language recognition'. In: *2020 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, pp. 186–190 (cit. on p. 7).

NumPy (2009). *NumPy*. URL: https://numpy.org/ (cit. on p. 20).

Pandiya, Mohit, Sayonee Dassani and P Mangalraj (2020). 'Analysis of deep learning architectures for object detection-a critical review'. In: *2020 IEEE-HYDCON*, pp. 1–6 (cit. on p. 34).

ProductPlan (n.d.). *Sprint | Agile Product Management | Definition and Overview*. URL: https://www.productplan.com/glossary/sprint/ (cit. on pp. vi, 14).

Python (2019). *Python*. URL: https://www.mybib.com/tools/ieee-citation-generator (cit. on p. 19).

Raschka, Sebastian, Joshua Patterson and Corey Nolet (2020). 'Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence'. In: *Information* 11.4, p. 193 (cit. on p. 34).

Rastgoo, Razieh, Kourosh Kiani and Sergio Escalera (2021). 'Sign language recognition: A deep survey'. In: *Expert Systems with Applications* 164, p. 113794 (cit. on pp. 2, 5).

– (2022). 'Real-time isolated hand sign language recognition using deep networks and SVD'. In: *Journal of Ambient Intelligence and Humanized Computing* 13.1, pp. 591–611 (cit. on p. 6).

Rehman, Anwar Ur et al. (2019). 'A hybrid CNN-LSTM model for improving accuracy of movie reviews sentiment analysis'. In: *Multimedia Tools and Applications* 78, pp. 26597–26613 (cit. on p. 8).

Saabith, S, T Vinothraj and M Fareez (2021). 'A review on Python libraries and Ides for Data Science'. In: *Int. J. Res. Eng. Sci* 9.11, pp. 36–53 (cit. on p. 33).

Sabeenian, RS, S Sai Bharathwaj and M Mohamed Aadhil (2020). 'Sign language recognition using deep learning and computer vision'. In: *J. Adv. Res. Dyn. Control Syst* 12.5, pp. 964–968 (cit. on p. 5).

Salehin, Imrus and Dae-Ki Kang (2023). 'A Review on Dropout Regularization Approaches for Deep Neural Networks within the Scholarly Domain'. In: *Electronics* 12.14, p. 3106 (cit. on p. 11).

Sathya, Ramadass, Annamma Abraham et al. (2013). 'Comparison of supervised and unsupervised learning algorithms for pattern classification'. In: *International Journal of Advanced Research in Artificial Intelligence* 2.2, pp. 34–38 (cit. on p. 7).

scikit-learn (2019). *scikit-learn: Machine Learning in Python*. URL: https://scikit-learn.org/stable/ (cit. on p. 20).

Sharma, Shikhar and Krishan Kumar (2021). 'ASL-3DCNN: American sign language recognition technique using 3-D convolutional neural networks'. In: *Multimedia Tools and Applications* 80.17, pp. 26319–26331 (cit. on p. 6).

Shorten, Connor and Taghi M Khoshgoftaar (2019). 'A survey on image data augmentation for deep learning'. In: *Journal of big data* 6.1, pp. 1–48 (cit. on p. 10).

Singhto, Wantana and Nuttaporn Phakdee (2016). 'Adopting a combination of Scrum and Waterfall methodologies in developing Tailor-made SaaS products for Thai Service and manufacturing SMEs'. In: *2016 International Computer Science and Engineering Conference (ICSEC)*. IEEE, pp. 1–6 (cit. on p. 13).

Staudemeyer, Ralf C and Eric Rothstein Morris (2019). 'Understanding LSTM–a tutorial into long short-term memory recurrent neural networks'. In: *arXiv preprint arXiv:1909.09586* (cit. on p. 8).

Streamlit (n.d.). *Streamlit: The fastest way to build data apps*. URL: https://streamlit.io/ (cit. on p. 20).

Team K (n.d.). *Keras documentation: TimeDistributed layer*. URL: https://keras.io/api/layers/recurrent_layers/time_distributed/. (cit. on p. 10).

TensorFlow (2019). *TensorFlow*. URL: https://www.tensorflow.org/ (cit. on p. 20).

Tiwari, Ashish (2022). 'Artificial Intelligence and Machine Learning for EDGE Computing'. In: Academic Press, pp. 23–32 (cit. on p. 42).

Turk, Dan, Robert France and Bernhard Rumpe (2002). 'Limitations of agile software processes'. In: *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2002)*. Citeseer, pp. 43–46 (cit. on p. 13).

Vafeiadis, Thanasis et al. (2022). 'A comparison of 2DCNN network architectures and boosting techniques for regression-based textile whiteness estimation'. In: *Simulation Modelling Practice and Theory* 114, p. 102400 (cit. on p. 8).

Wadhawan, Ankita and Parteek Kumar (2021). 'Sign language recognition systems: A decade systematic literature review'. In: *Archives of Computational Methods in Engineering* 28, pp. 785–813 (cit. on p. 2).

WHO (2019). *Deafness and hearing loss*. URL: https://www.who.int/health-topics/hearing-loss#tab=tab (cit. on p. 1).

Yang, Su and Qing Zhu (2017). 'Continuous Chinese sign language recognition with CNN-LSTM'. In: *Ninth international conference on digital image processing (ICDIP 2017)*. Vol. 10420. SPIE, pp. 83–89 (cit. on p. 6).

Ying, Xue (2019). 'An overview of overfitting and its solutions'. In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing, p. 022022 (cit. on p. 11).

Yu, Juezhao et al. (2020). '2D CNN versus 3D CNN for false-positive reduction in lung cancer screening'. In: *Journal of Medical Imaging* 7.5, pp. 051202–051202 (cit. on p. 25).

Zhang, Liang et al. (2017). 'Learning spatiotemporal features using 3dcnn and convolutional lstm for gesture recognition'. In: *Proceedings of the IEEE international conference on computer vision workshops*, pp. 3120–3128 (cit. on p. 8).