University of Technology, Sydney
Faculty of Engineering and Information Technology

Applied Machine Learning in Natural Language Processing and Video Processing Techniques for Sign Languages

by

Thomas Gallagher

Student Number: 13276264
Project ID: SPR-22-09273
Major: Data Engineering Major

Academic Advisor: Thuy Pham

A 12 Credit Point Project submitted in partial fulfilment of the requirement for the Degree of Bachelor of Engineering
7 November 2022

ASSIGNMENT COVERSHEET

UTS: ENGINEERING & INFORMATION TECHNOLOGY

SUBJECT NUMBER & NAME	NAME OF STUDENT(s) (PRINT CLEARLY)	STUDENT ID(s)
41030 Engineering Capstone	Thomas Gallagher	13276264

STUDENT EMAIL	STUDENT CONTACT NUMBER
Thomas.gallagher@student.uts.edu.au	0434356978

NAME OF TUTOR	TUTORIAL GROUP	DUE DATE
Thuy Pham		07/11/2022

ASSESSMENT ITEM NUMBER & TITLE

Assessment Task 1: Final Report

☒ I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.
☒ I confirm that I have read, understood and followed the advice in the Subject Outline about assessment requirements.
☒ I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.

Declaration of originality: The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I understand that, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.

Statement of collaboration:

Signature of student(s)  **Date** 7/11/2022

✂----------

ASSIGNMENT RECEIPT

To be completed by the student if a receipt is required

SUBJECT NUMBER & NAME	NAME OF TUTOR

SIGNATURE OF TUTOR	RECEIVED DATE

STYLE GUIDE for ASSIGNMENT SUBMISSION

Before submitting an assignment, you should refer to the policies and guidelines set out in the following:

- FEIT Student Guide
- UTS Library - referencing
- HELPS - English and academic literacy support
- UTS GSU - coursework assessment policy and procedures

Unless your Subject Coordinator has indicated otherwise in the Subject Outline, you must follow the instructions below for submission of assignments in the Faculty of Engineering and Information Technology.

Writing style

It is usually best to write your initial draft in the default settings of your software without formatting. Use the following guides in your writing.

Purpose and audience: use the correct genre and language style expected for the particular task.

Language: use 'plain English' for all technical writing. More information about this language style can be found at www.plainenglish.co.uk/free-guides.html.

Use spelling and grammar software tools to check your writing. Edit your document.

Standards: always use:

- Australian spelling standards (Macquarie Dictionary)
- SI (International System of Units) units of measurement
- ISO (International Organisation for Standardisation) for writing dates and times for international documents. For example **yyyy-mm-dd** or **hh-mm-ss**. However, for most applications it is more helpful to present the date in full as **26 August 2016**.

Graphics and tables should:

- be numbered
- have an appropriate heading and/or caption
- be fully labelled
- be correctly referenced.

Presentation

Unless otherwise instructed, all assignment submissions should be **word processed** using spell-check and grammar-check software. Work should be well **edited** before submission. Use the following default settings:

Page setup: set margins at no less than 20mm all around.

Paper: print on A4 bond, double-spaced and preferably double-sided, left justified.

Font: use the software default style to provide consistency. The recommended style includes:

- 10-12 pt font
- consistent formatting with a limited number of fonts
- lines no more than 60 characters (use wider margins or columns if you need to make lines shorter)

Header should include:

- your name and student number
- the title of the paper or task.

Footer should include the page number and current date.

Cover sheet and statement of originality: all work submitted for assessment must be the original work of the student(s) submitting the work. A standard faculty cover sheet (see over) must be attached to the front of the submission. Any collaboration between the submitting student and others must be declared on the cover sheet.

Referencing

All sources of information used in the preparation of your submission must be acknowledged using the Harvard system of referencing. This includes all print, video, electronic sources.

Phrases, sentences or paragraphs taken verbatim from a source must be in quotation marks and the source(s) cited using both **in-text** referencing and a **reference list**.

Plagiarism is the failure to acknowledge sources of information. You should be fully aware of the meaning of plagiarism and its consequences both to your marks, position at the university and criminal liability. The plagiarism in your assignment submissions can be assessed both in hard copy and in soft copy through software such as Turnitin.

The UTS Library and UTS HELPS (web links above) provide extensive information for students on referencing correctly to support you in avoiding plagiarism.

Applied Machine Learning in Natural Language Processing and Video Processing Techniques for Sign Languages - (12cp)
Thomas Gallagher - SPR-22-09273

Supervisor: Thuy Pham
Major: Data Engineering Major

Objective:
Determine if a software model can be trained to accurately translate sign language videos of a specific language.

Machine learning for video processing is an area of study that has been explored quite extensively. However, much less attention has been given to applying machine learning techniques to the area of sign language. Automatic Sign Language Recognition (ASLR) presents a number of challenges, including differentiating between sign languages for each sign, identifying the completion of a particular sign and start of the next sign in a video, and combining facial expressions with hand gestures and body movement to recognise signs. Sign language is a crucial tool for communication between hearing-impaired people and hearing persons, and any development towards improving ASLR is a great benefit to this area.

In this capstone project, I present two machine learning methods for accurate classification of sign instances. The first model utilises MATLAB, combining a pre-trained convolutional neural network image classification model with a Long Short-Term Memory network to create a network for video classification. The second model utilises Python, developing a Keras-based sequential convolutional neural network that classifies sign videos using a single-frame CNN approach.

Both models were trained and tested with ten classes found in the American Sign Language Linguistic Research Project (ASLLRP) dataset. The dataset provides hundreds of instances of short individual sign instance videos, and the largest classes in this dataset were chosen for the development of the models.

The results of both models are detailed in this work, along with a discussion regarding the validity of findings, possible future work, and overall contribution of the project towards determining if a software model can be trained to accurately classify signs of a specific language.

Introduction

Automatic Sign Language Recognition (ASLR) refers to the process of identifying the meaning of sign language automatically, with the ability for translation into spoken language. As Adeyanjua, Bello and Adegboye (2021) describe, sign language is a crucial tool for communication between hearing-impaired people and hearing persons, and with the large amount of varying sign languages that all have unique motion positions, body positions and hand shapes, it is a complicated process to develop ASLR.

The first challenge involving ASLR is being able to differentiate types of sign languages. Sign language is not a universal language, and is also independent of spoken languages (Swisher, 1988). Some languages are distinct enough from each other that distinguishing languages using ASLR can be less challenging, such as British Sign Language (BSL) and American Sign Language (ASL). Figure A below showcases this, demonstrating how BSL uses two hands to sign letters, whereas ASL uses only one.

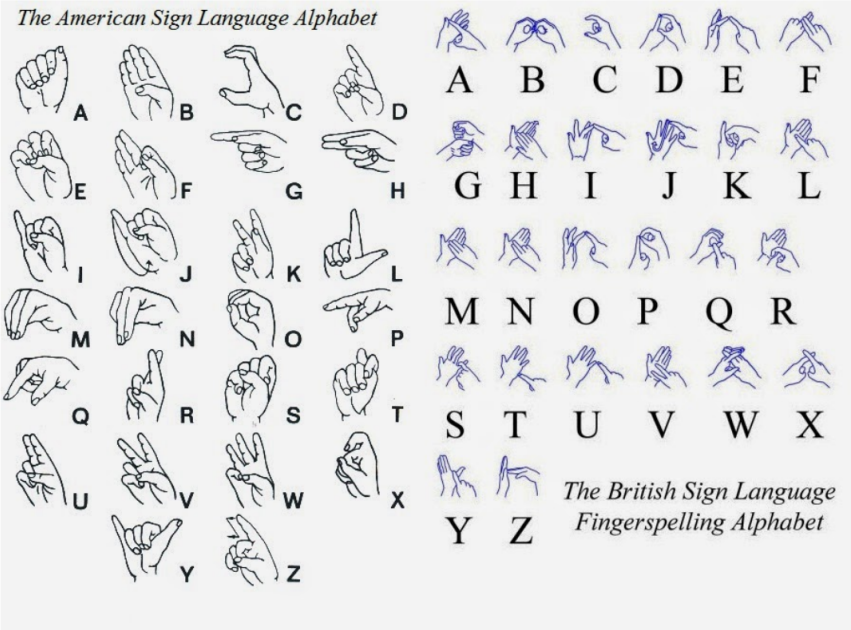


Figure A: A comparison of the alphabets of ASL and BSL (Vallin, 2017).

In other cases, sign languages can be so similar to one another that they are considered dialects. BANZL is a sign language family that incorporates BSL, Australian Sign Language (Auslan) and New Zealand Sign Language (NZSL). These three types of sign languages share most signs, also using the same manual alphabet and grammar (Sign Community, 2013). Being able to automate differentiation between sign language types such as these is a challenging process.

The second challenge involving ASLR is the actual recognition of signs in video. Signers communicate not only through their hands, but also through their facial expressions and head

movements (Elliot & Jacobs, 2013), which need to be combined by recognition technology to produce an accurate interpretation. To add to this challenge, the appearance of a new sign in the dynamic sign language process is influenced by the previous sign. Furthermore, these signs can look quite different from one signer to another. Figure B below demonstrates this, comparing snapshots of different signers for various pointing gestures. As can be seen, not only are these pointing gestures quite similar to each other, the same gesture varies between signers.

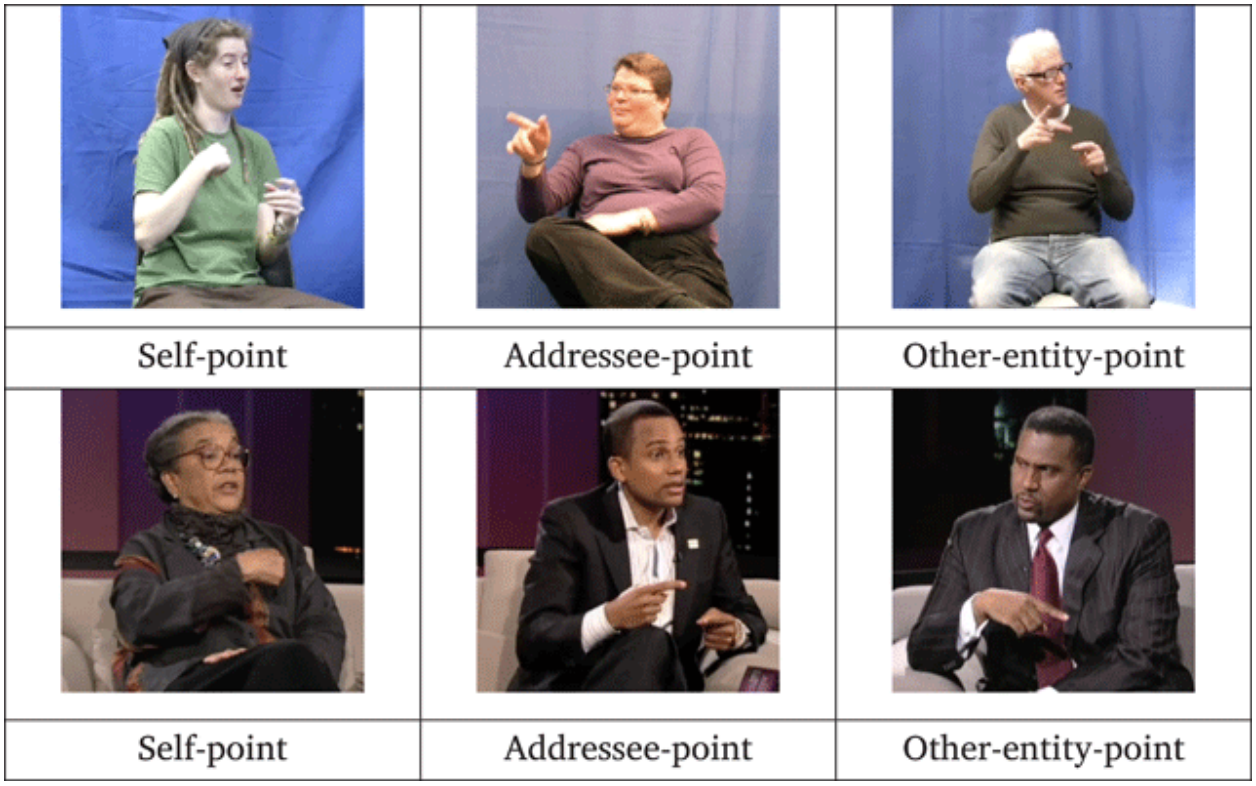


Figure B: A comparison of different speakers performing pointing gestures (Fenlon et al., 2019).

With these challenges in mind, this project has attempted to address the following research question: Can a software model(s) be trained to accurately translate sign language videos of a specific language?

Literature Review

This research project was developed utilising the existing ideas and work towards achieving ASLR made by Albanie et al. (2021). Albanie et al. introduce a scalable method of sign language data collection in continuous videos, utilising weakly-aligned subtitles in broadcast footage along with keyword spotting to isolate sign-instances. Through this process, they have isolated over 100 hours of signing, creating an accessibility dataset with the intention to support ASLR research tasks. Through further experimentation with this data, Albanie et al. determine that the contribution of mouth and face add to the accuracy of the model, reinforcing Elliot & Jacobs' idea that more than just hand signals are required for accurate ASLR.

Whilst this database was initially intended to be utilised for this research task, there are a number of drawbacks and limitations that comes with using this data. Firstly, a potential issue with this dataset and research as a whole is the assumption of a one-to-one mapping of sign to spoken word. For a machine learning algorithm to work optimally, the training data should be fairly consistent, as performance of the model greatly depends on the quality of this data (Fenza et al., 2021). In sign language however, a particular sign may be a combination of words, and can vary depending on the context, which in turn can greatly impact the goal of achieving ASLR that Albanie et al. are working towards through their dataset. The dataset also focuses on leveraging mouthing cues to create the dataset, and Albanie et. al themselves point out that whilst mouthing is a strong signal, it cannot annotate all data (ie. signers do not mouth continuously). Furthermore, this annotated data is skewed towards words that are simple to mouth, which again points to concerns of data consistency as well as integrity. In summary, attempting to expand upon the work of Albanie et al. is not without its flaws, but was to act as the initial starting point for this research project.

Adeyanjua, Bello and Adegboye (2021) describe a number of explored machine learning techniques for sign language recognition. They identify the different thresholding techniques explored by various researchers, such as Otsu's thresholding algorithm to segment the hand region used by Rahim et al. (2020) and Pansare et al. (2012), contrasted against skin colour segmentation as a multilevel thresholding technique for hand segmentation, a technique explored by Razmjooy et al. (2021). Adeyanjua, Bello and Adegboye provide an analysis of a number of different proposed systems in the context of sign language recognition, ultimately concluding that a perfect sign language recognition system is an open problem. They note that future research should be more attentive to studying the face region of non-manual signs, coordinating facial expression with hand gestures and body movement, sentence recognition rather than single words or alphabet, and implementation of proposed models that result in practical applications rather than halting at the research and prototype stage. From these findings, it is clear that the ASLR space has valuable potential to be explored, and that the proposed objective of this research project is relevant and significant to this space.

Methodology

Research Design

The design process for this project followed the system development life cycle (SDLC) model, a popular framework for software development allowing products to be designed, developed and deployed with efficiency and quality (Preston, 2022). Preston describes the 7 phases of this model; planning, analysis, design, development, testing, integration and maintenance. As this product is not at the stage of reaching a market, the maintenance stage is absent for this project. With the above phases in mind, the following milestones were defined:

Milestone 1: Preparation
Milestone 2: Software Design
Milestone 3: Software Development
Milestone 4: Analysis

For the most part, the above milestones were completed in order, however, due to issues involving pivoting of datasets and changing of scope, aspects within these milestones evolved and overlapped as the project went on. The preparation stage involved the collection of sign language datasets and identification of software requirements the project needed. The software design stage involved the research of potential machine learning techniques that were applicable to the project and installation of the tools required to perform these techniques. The software development stage included the development and training of the model, and the analysis stage involved the evaluation and discussion of the model results. Whilst the SDLC model showcases the general pattern of work that I followed, the milestones enabled me to measure physical results over time.

Data Collection

In the context of this research proposal, data refers to the sign language datasets to be used for training and testing an ASLR model. This research project focused on utilising existing sign language datasets to develop an ASLR model, and as such did not concentrate on the actual process of data collection. Initially, the primary data to be used was the BBC-Oxford British Sign Language (BOBSL) dataset created by Albanie et al. However, this dataset was in the form of long BBC broadcast footage of signers, accompanied by files containing timestamps for specific sentences and words. As I wanted the scope for this project to be on the development of a machine learning model rather than the creation of a sign language dataset, I chose to explore alternative sign language datasets rather than commit to utilising the BOBSL dataset.

BOBSL is not the only sign language dataset available to develop a machine learning model, and others were identified that also have the potential for model development. Kadous (2002) provides a dataset of Auslan signs, however, this dataset focuses on tracking hand motion. As was discussed in the literature review, one of the focal areas for future work is coordinating

facial expression with hand gestures and body movement, which is the aspect that I decided to centre on when choosing the dataset. Another explored option was the American Sign Language Lexicon Video Dataset (ASLLVD) created by Athitsos et al. (2008), which contains videos of thousands of distinct ASL signs along with annotations. However, the format of these videos is similar to the BOBSL dataset, and hence using this dataset was also not chosen as it would involve expanding the scope of this project to also generating the sign language dataset.

The initial dataset tested and trained was Auslan SignBank (Auslan Signbank, n.d.), a language resource site for Australian Sign Language (Auslan). This dataset was chosen as the videos contain the entire person performing the sign (including face and hands), and showcases all signs in the Auslan dictionary. Furthermore, each video runs for 1-5 seconds, which makes the dataset ideal for training and testing a machine learning model. Figure C below showcases a snippet of the sign ‘abdomen’ from the database.

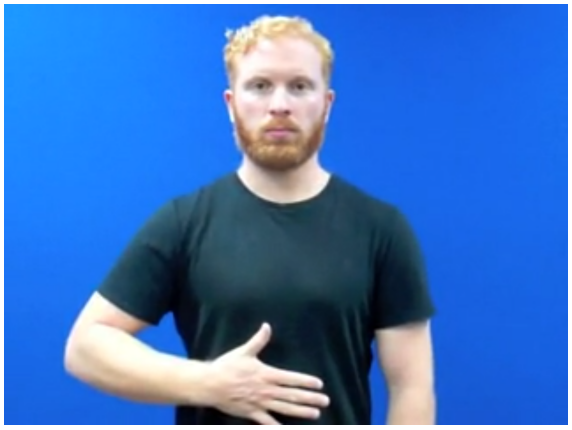


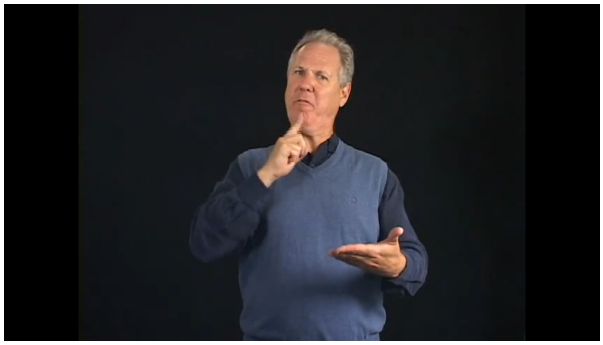
Figure C: A snippet of the sign ‘abdomen’ from the Auslan SignBank database.

Whilst the Auslan SignBank was excellent for breadth of signs, it lacked the required depth needed for training a suitable machine learning model. Majority of signs (classes) contained fewer than 20 sign videos, resulting in poor predictions from the trained classifier. As such, the final dataset explored and utilised for this project was the American Sign Language Linguistic Research Project (ASLLRP) dataset (Boston University, 2022). Similar to Auslan Signbank, the ASLLRP provides video clips of instances of sign words, useful for training a machine learning model. Furthermore, the number of video instances per class is much more than those in the Auslan Signbank. However, this is not the case for all signs, with most classes containing less than 10 instances, but large classes containing 50-200. With the research question in mind, as well as the recommendation of pursuing the cooperating of both facial expression and hand gesturing made by Adeyanjua, Bello and Adegboye (which ASLLRP accomplishes), it was decided that 10 of the largest classes would be chosen from the ASLLRP dataset to be used for development of a machine learning model.

Table 1 below showcases the 10 chosen classes, along with a snippet of a training sample taken from one of the class datasets. As can be seen, speakers are not limited to a specific class, which is useful for training a model, as it means that characteristics including face, gender or hairstyle cannot be used by the model to ascertain the class. From the initial inspection of

each class, it was proposed that the model would have a fairly varied accuracy between classes, as some classes are quite distinct from others (such as ‘car’), whilst others are much more similar (such as ‘deaf’, ‘future’, ‘must’ and ‘who’). Although this project does not include all American signs, observing the predictions of similar signs within the 10 classes can help indicate the potential of a model that has been trained on a much wider class dataset, where certain classes would be even more similar to one another.

car


deaf

finish

friend

future

go-out

 must	 now
 who	 why

Table 1: Snippets of a sample training video from each class.

Although some of these classes contained over 100 samples, there were other classes that contained 70-80 videos. As explained by Elrahman & Abraham (2013), class imbalance degrades the overall performance of machine learning techniques, resulting in a classifier that is often biassed towards the majority class. As such, the majority classes were downsampled to match the minority classes. Splitting the dataset in a test to train ratio of 85:15, the result of this is 10 classes each with 60 training samples and 10 test samples.

Data Processing

MATLAB Classifier

The first classifier trained on the Auslan Signbank videos and ASLLRP videos was a deep learning network adapted from the ideas of Mathworks (2018), describing a method of combining a pretrained image classification model with a Long Short-Term Memory (LSTM) network to create a network for video classification. The pretrained network utilised for this project is GoogLeNet, a model that has been trained over 1 million images in the ImageNet database for the ImageNet Large-Scale Visual Recognition Challenge. However, sign language actions are not a part of this database, and the GoogLeNet network is only used to extract features vectors from the videos. Figure D below encapsulates this process, showcasing that the convolutional network extracts the activations for each video frame inputted into the

network. The output of the activations function on the final pooling layer of the GoogLeNet network act as feature vectors for the video.

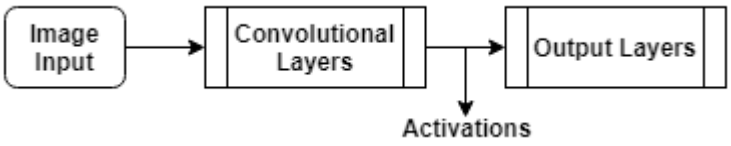


Figure D: A diagram of the flow of data through the GoogleNet network (Mathworks, 2018).

The LSTM network is created to classify the above extracted feature vectors representing the videos. The LSTM network architecture involves a sequence input layer, bidirectional LSTM layer containing 2000 hidden units followed by a dropout layer, fully connected layer, softmax layer, and a classification layer. Figure E below details the creation of these layers in MATLAB, which are later used as a parameter when training the network.

```
numFeatures = size(sequencesTrain{1},1);
numClasses = numel(categories(labelsTrain));

layers = [
    sequenceInputLayer(numFeatures,'Name','sequence')
    bilstmLayer(2000,'OutputMode','last','Name','bilstm')
    dropoutLayer(0.5,'Name','drop')
    fullyConnectedLayer(numClasses,'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','classification')];
```

Figure E: The MATLAB code that generates the LSTM network layers.

The entire combined network is demonstrated in Figure F below. To summarise, the input image sequences (i.e. individual frames of a video) are inputted to the network, which are then transformed into a series of feature vectors within the sequence folding layer, convolutional layers, sequence unfolding layer and flatten layer. The resulting vector sequences are fed into the LSTM layers followed by the output layers to classify the input video.

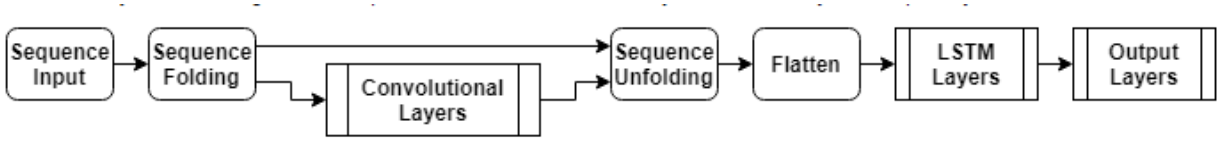


Figure F: A diagram of the network architecture for the video classifier (Mathworks, 2018).

Table 2 below indicates the training options specified for the MATLAB classifier. Appendix A showcases the entire MATLAB script for this generated classifier, including the classification of the test dataset.

Parameter	Model Option
Size of mini-batch	16
Initial learning rate	1e-4
Gradient threshold	2
Option for data shuffling	Every epoch

Table 2: The training options specified for the LSTM network.

Python Classifier

The second classifier trained was a single-frame convolutional neural network (CNN) video classifier using Keras, an open-source software library providing an interface in Python for neural networks. Unlike the previous classifier, this neural network was created in Python, however, a similar process of data processing, network construction and model evaluation was performed.

Figure G below details the reading and preprocessing of the dataset for the classifier. All videos in each class are sent as input into the 'frames_extraction' function, with randomly selected video frames being added to the 'features' variable.

```
def create_dataset():
    # extract videos in each class and return features and labels of each video
    temp_features = []
    features = []
    labels = []

    for class_index, class_name in enumerate(classes_list):
        print(f'Extracting Data of Class: {class_name}')
        files_list = os.listdir(os.path.join(dataset_directory, class_name))

        for file_name in files_list:
            video_file_path = os.path.join(dataset_directory, class_name, file_name)
            frames = frames_extraction(video_file_path)
            temp_features.extend(frames)

        features.extend(random.sample(temp_features, 200))
        labels.extend([class_index] * max_images_per_class)
        temp_features.clear()

    features = np.asarray(features)
    labels = np.array(labels)

    return features, labels
```

Figure G: A screenshot of the 'create_dataset' function, which iterates through all training images for each class and returns a list of features and associated labels.

A sequential CNN is created, containing convolutional layers, normalisation layers, pooling layers, and dense layers. Figure H showcases the resulting structure of the generated model. This model is then used to make predictions on the test set. This model is created utilising Keras, an open-source Python library designed for deep learning frameworks (Keras, 2022).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 64)        1792

 conv2d_1 (Conv2D)           (None, 60, 60, 64)        36928

 batch_normalization (BatchN  (None, 60, 60, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 30, 30, 64)       0
 )

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               16640

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 59,210
Trainable params: 58,570
Non-trainable params: 640
```

Figure H: A view of the classifier model architecture.

The above model generates a prediction for each individual frame in the input video. Because the test videos are quite short and contain few leading/trailing frames where the speaker is not performing the sign, a single-frame CNN approach is used. This method involves running the image classification model on every frame in the input video and averaging all individual probabilities for each class. The class with the highest average probability is returned as the predicted result. Appendix B showcases the entire Python code for this classifier.

Analysis

MATLAB Classifier on Auslan Signbank

The MATLAB model described above was first trained and tested on the Auslan Signbank dataset. The poor results demonstrated that the dataset was not deep enough to be used for training a machine learning model. Table 3 details the results of 5 of the test videos along with the top 5 predictions of the model. As can be seen, the majority of top predictions are of the 'animal' class, and not a single top-5 prediction is accurate for the class. This issue was present in the entire test dataset, and as such it was suspected that the problem was due to an imbalanced dataset. As Brownlee (2020) describes, imbalanced data can result in a poor predictive performance as the minority class is valued less. However, this was not the case for the Auslan Signbank dataset, and so it was surmised that the model was unable to extract meaningful features from each class using the limited training data.

True Class	Top Prediction	Second Prediction	Third Prediction	Fourth Prediction	Fifth Prediction
Abdomen	Animal	Cheap	Character	Dormitory	Cheerleader
Deaf	Algebra	Animal	Death	Bug	Goat
Laugh	Animal	Airplane	Accident	Death	Chat
Read	Animal	Analyse	Coconut	Amazing	Goat
Wait	Analyse	Animal	Airplane	Algebra	Death

Table 3: The top 5 predictions of the MATLAB classifier on 5 test samples from Auslan Signbank.

MATLAB Classifier on ASLLRP

Due to the results using the Auslan Signbank dataset, the same MATLAB classifier was trained using the ASLLRP dataset instead. Figure I showcases the training progress of the classifier over each epoch. As can be seen, the model was trained for 30 epochs, resulting in a validation accuracy of 75.56%.

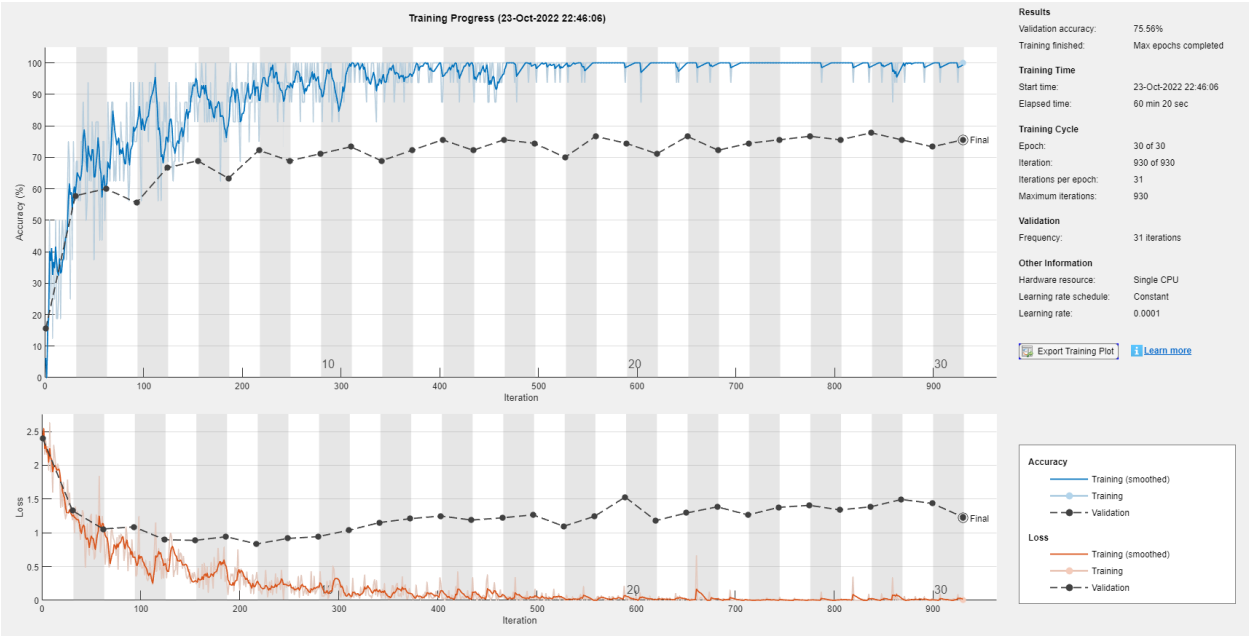


Figure I: An illustration of the training progress of the MATLAB classifier on the ASLLRP dataset.

Table 4 details the results of the trained model. The classifier performed much better on the data, resulting in an accuracy of 81%. When analysing the training dataset, it was hypothesised that the model would struggle the most with distinguishing between the classes ‘deaf’, ‘future’, ‘must’ and ‘who’, as these are quite similar in action to each other. As can be seen, although ‘deaf’ achieved 100% accuracy, the other 3 classes mentioned were 3 of the lowest scoring classes, which gives some merit to the above hypothesis. Overall, the results show that a software model can indeed be trained to accurately translate sign language videos.

	Predicted Class										
True Class	Class Name	Car	Deaf	Finish	Friend	Future	Go-out	Must	Now	Who	Why
	Car	9						1			
	Deaf		10								
	Finish			9				1			
	Friend				10						
	Future		2	1		5				1	1
	Go-out						8	1			1
	Must					1	2	7			
	Now			1				2	7		
	Who		3							7	
	Why									1	9

Table 4: The results of the MATLAB classifier on the ASLLRP test data.

Python Classifier on ASLLRP

Although the above classifier was very successful, the model uses transfer learning to retrain the pre-trained GoogLNet model to classify the images. I wanted to observe if creating a network from scratch that is trained purely on the sign language videos would produce better results, and hence the Python classifier described above was created and tested on the same ASLLRP dataset. Table 5 details the results of the model, which achieved an accuracy of 85%, slightly higher than the MATLAB model. Interestingly, the model struggled the most with the sign ‘finish’, predicting 40% of the test videos of this class as ‘must’, which appears to bear little relation to ‘finish’ when comparing the class videos. However, for the most part, the Python model performed quite similarly to the MATLAB model, and was able to successfully discern the features that separate each class.

		Predicted Class									
True Class	Class Name	Car	Deaf	Finish	Friend	Future	Go-out	Must	Now	Who	Why
	Car	10									
	Deaf		10								
	Finish			6				4			
	Friend				10						
	Future				1	7				2	
	Go-out						8			1	1
	Must					1	1	8			
	Now							2	8		
	Who		2							8	
	Why						1				9

Table 5: The results of the Python classifier on the ASLLRP test data.

Discussion

Validity of Findings

Whilst I initially intended to utilise all signs in a particular sign language, I was only able to implement 10 signs from the ASLLRP dataset, as these signs contained the most number of videos. Whilst the high accuracy of the results show that these particular signs are classifiable for a trained machine learning model, it is possible that as more classes are introduced, the accuracy of the model decreases. A study by Novianti et al. (2015) showed that one of the factors that affect the accuracy of a class prediction model in gene expression data is the number of differentially expressed genes. If this concept is also the case for a sign language model, then given the vast amount of words in the ASL dictionary, it is likely that the results of the model would be vastly worse.

Although the number of training data needed for machine learning varies drastically from project to project, it is generally considered that the more data one has, the better (Dorfman, 2022). Whilst the classes used from the ASLLRP dataset were chosen based on the amount of videos they contained, I initially felt that there was not enough data for a model to train on and extract meaningful and distinctive features amongst the classes. However, even with this limited data, both models performed very well. The results definitely answer the initial research question, and I expect that with a much larger dataset, an accuracy of >95% is definitely possible.

Furthermore, whilst both classifiers employ CNN architecture, they do so using different methods. The MATLAB classifier combines a pre-trained CNN with a LSTM network, whilst a sequential CNN is built from scratch in the Python classifier. However, both methods achieve similarly high results, proving that the validity of the findings are not model-dependent.

Challenges

Changing Datasets

I had initially intended to make use of the BOBSL dataset containing hours of sign videos. However, the format of the dataset and complexity of the assisting classifiers produced by Albanie et al. affected my decision to utilise a dataset that was already in a format capable of being input for a model and focus on the model development rather than the dataset development. The Auslan Signbank was initially chosen, but due to the results, I again chose to switch datasets to the ASLLRP dataset. At this stage in the project, I felt that whilst the dataset was not the most optimal dataset for this project, I should focus on the model development, and hence reduced the scope of the classes rather than switching datasets again.

Change of Scope

As was mentioned, the scope of the model changed over the duration of the project lifetime. The BOBSL dataset that was initially downloaded contains timestamps of both words and sentences, so I had intended to explore both of these activities, such as identifying individual words in a sentence. However, as the dataset changed, the new datasets only contained smaller clips of individual signs. Furthermore, the machine learning techniques that were explored extracted each frame from the videos, before applying image classification techniques on those frames. This means that the less frames the video has, the faster the machine learning model can be trained and tested. After some consideration, I elected to focus purely on individual sign classification, rather than exploring sentence construction and deconstruction.

The ASLLRP dataset chosen contains over 23,000 different isolated sign videos. However, the majority of words in the ASLLRP sign bank contain fewer than 15 video instances. From the results of the MATLAB model that had been trained on the Auslan Signbank dataset, it was clear that this amount of data for training would not be enough. As was mentioned above, at this stage in the project timeline I felt that it was more important to focus on the development of the machine learning model rather than spend time switching datasets again. As such, I chose to cut down on the number of classes from the ASLLRP, identifying and using only the 10 largest classes in the dataset. Although this decision has led to discussion regarding the validity of the findings, I believe it has allowed me to demonstrate that a software model can be trained to accurately translate sign language videos of a specific language, which aligns with the ultimate goal of this research project.

Future Work

I believe that the project has a lot of potential in the realm of ASLR, with the most obvious next step being the inclusion of more classes for the models. Comparing the results of the above generated models with models that predict 20-30 classes would help to determine the validity of the above findings, and establish how the utilised machine learning processes fare with a larger scope. As was mentioned, I was not satisfied with the final chosen dataset, and am confident that a larger dataset would improve the model's accuracy. Another possible future work is to focus on this conjecture and locate/create a dataset that is much deeper than the utilised ASLLRP dataset.

Reflection

In hindsight, I believe it could have been more beneficial to focus further on the BOBSL dataset and accompanying Python code produced by Albanie et al. At the time, I was worried about not having time to work on the development of the models, but ended up spending a significant amount of time changing datasets. If I had instead continued to explore the BOBSL dataset and created a dataset of word instances using the BBC footage and supplied timestamps, the project could have potentially concluded with a model that is trained on thousands of sign videos with the capability of classifying far more classes.

I also believe I could have better communicated with my supervisor throughout the course of this research project. My supervisor was very encouraging and knowledgeable on the subject matter, however, I did not convey my progress as often as I believe I should have, which I believe was a detriment to the project. I think if I had discussed the issues and ideas I had throughout the project with my supervisor, I could have better aligned my scope and work with my proposed research objective.

During the project, my supervisor let me know that another student was completing the same project, but was in the research preparation phase. My supervisor offered to introduce me to this student so that we could discuss the topic and possibly collaborate on a final product. I was happy to discuss this with the student, and after some back-and-forth emailing, asked them to have a Teams meeting with me to discuss my progress and whether they wanted to work with me. Although I set up a meeting time, the member was not available, and didn't respond when I attempted to set up another meeting time. Although I am unsure how this collaborative project would have looked, I was a little disappointed that the student gave no indication of whether they wanted to work on this project.

As was discussed, I was worried that the ASLLRP dataset did not contain enough data necessary for the trained classifiers. However, I was pleasantly surprised with the results of both the MATLAB and Python classifiers, which has made me optimistic about the future of ASLR. Overall, I found the project to be both challenging and rewarding to work on. Having the understanding that this project was contributing to the development of sign language recognition was exciting and motivating for me, enabling me to continue to work to the best of my ability. As difficult as the research project was, I am quite pleased with the outcome and have learnt a remarkable amount regarding sign language recognition and applying machine learning techniques to sign language video.

Conclusion

Automatic sign language recognition is a challenging problem in the world, and whilst there have been many steps towards achieving a solution, more research and progress is needed to completely solve the issue. This research project has sought to answer if a software model(s) can be trained to accurately translate sign language videos of a specific language. Two distinct CNN models were trained to classify videos of 10 distinct ASL signs, achieving resulting accuracies of 81% and 84% respectively. Although the results of the model do not represent a complete solution to all signs in the ASL dictionary, the success of the results with a limited dataset are a positive indication that such a classifier is possible. Whilst the research project does not solve the problem of classifying sign language in video, I believe it helps to highlight that such a problem can be solved with machine learning techniques, and that automatic sign language recognition is an attainable target for future development.

References

Adeyanju, I.A, Bello, O.O & Adegboye, M.A. (2021). Machine learning methods for sign language recognition: A critical review and analysis. *Intelligent Systems with Applications, 12*(200056). https://doi.org/10.1016/j.iswa.2021.200056.

Albanie, S., Varol, G., Momeni, L., Triantafyllos Afouras, T., Chung, J.S., Fox, N. & Zisserman, A. (2021, October 13). *BSL-1K: Scaling up co-articulated sign language recognition using mouthing cues*. https://doi.org/10.48550/arXiv.2007.12131.

Anwar, T. (2021, March 8). *Introduction to Video Classification and Human Activity Recognition*. https://learnopencv.com/introduction-to-video-classification-and-human-activity-recognition/.

Athitsos, V., Neidle, C., Sclaroff, S., Nash, J., Stefan, A., Yuan, Q., & Thangali, A. (2008, June). The American Sign Language Lexicon Video Dataset, *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (pp. 1-8). https://doi.org/10.1109/CVPRW.2008.4563181.

Auslan Signbank (n.d.) *Welcome to Auslan Signbank*. [Data set]. https://auslan.org.au/.

Boston University (2022). *American Sign Language Linguistic Research Project*. [Data set]. https://www.bu.edu/asllrp/.

Brownlee, J. (2020, January 14) *A Gentle Introduction to Imbalanced Classification*. https://machinelearningmastery.com/what-is-imbalanced-classification/.

Dorfman, E. (2022, March 25). *How Much Data Is Required for Machine Learning?*. https://postindustria.com/how-much-data-is-required-for-machine-learning/.

Elliott, E.A., & Jacobs, A.M. (2013). Facial expressions, emotions, and sign languages. *Frontiers in psychology*, 4(115). https://doi.org/10.3389/fpsyg.2013.00115.

Elrahman, S.M., & Abraham, A. (2013). A Review of Class Imbalance Problem. *Journal of Network and Innovative Computing,* 1, 332-340.

Fenlon, J., Cooperrider, K., Keane, J., Brentari, D. & Goldin-Meadow, S., (2019). Comparing sign language and gesture: Insights from pointing. *Glossa: a journal of general linguistics,* 4(1). https://doi.org/10.5334/gjgl.499.

Fenza, G., Gallo, M., Loia, V., Orciuoli, F. & Herrera-Viedma, E. (2021, July). Data set quality in Machine Learning: Consistency measure based on Group Decision Making. *Applied Soft Computing*, *106*. https://doi.org/10.1016/j.asoc.2021.107366.

Kadous, M. W. (2002, October). *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series* (PhD Thesis), School of Computer Science and Engineering, University of New South Wales.

Keras (2022). *Keras*. https://keras.io/.

Mathworks (2018). *Classify Videos Using Deep Learning*. https://au.mathworks.com/help/deeplearning/ug/classify-videos-using-deep-learning.html

Novianti, P.W, Jong V.L, Roes, K.C. & Eijkemans, M.J. (2015, June 21). Factors affecting the accuracy of a class prediction model in gene expression data. *BCM Bioinformatics*, 16(199), https://doi.org/10.1186/s12859-015-0610-4.

Pansare J. R., Gawande, S.H., & Ingle, M. (2012). Real-time static hand gesture recognition for American sign language (ASL) in complex background. *Journal of Signal and Information Processing*, 3(3), 364–367. https://doi.org/10.4236/jsip.2012.33047.

Preston, M. (2022, January 21). *System Development life Cycle Guide*. CloudDefense. https://www.clouddefense.ai/blog/system-development-life-cycle.

Rahim, M A, Shin, J., & Yun, K. S (2020). Hand gesture-based sign alphabet recognition and sentence interpretation using a convolutional neural network. *Annals of Emerging Technologies in Computing*, 4(4), 20–27. https://doi.org/10.33166/AETiC.2020.04.003.

Razmjooy, N., Razmjooy, S., Vahedi, Z., Estrela, V.V, & de Oliveira, G. G. (2021). Skin color segmentation based on artificial neural network improved by a modified grasshopper optimization algorithm. *Lecture Notes in Electrical Engineering*, 696, 169-185. https://doi.org/10.1007/978-3-030-56689-0_9.

Sign Community (2013). *BANZSL.* https://www.signcommunity.org.uk/banzsl.html.

Suharjitoa, Ricky Anderson, R., Wiryanab, F., Ariesta, M.C. & Kusumaa, G.P. (2017, October 13). Sign Language Recognition Application Systems for Deaf-Mute People: A Review Based on Input-Process-Output. *2nd International Conference on Computer Science and Computational Intelligence 2017*, 116, 441-448. https://doi.org/10.1016/j.procs.2017.10.028.

Swisher, V.M. (1988, December). Similarities and Differences between Spoken Languages and Natural Sign Languages. *Applied Linguistics*, 9(4), 343–356. https://doi.org/10.1093/applin/9.4.343.

Appendix

Appendix A: MATLAB code for the MATLAB Classifier

```
%% Classify Videos using a pretrained GoogLeNet CNN combined with an LSTM
network
%% Load Pretrained Convolutional Network
netCNN = googlenet;
%% Load Data
dataFolder = "D:\Uni\capstone\ASLLRP\train";
testFolder = "D:\Uni\capstone\ASLLRP\test";
[files,labels] = hmdb51Files(dataFolder);
%%
% Read the first video and view the size and corresponding label of the video
idx = 1;
filename = files(idx);
video = readVideo(filename);
size(video)
labels(idx)
%%
% View the video
numFrames = size(video,4);
figure
for i = 1:numFrames
   frame = video(:,:,:,i);
   imshow(frame/255);
   drawnow
end
%% Convert Frames to Feature Vectors
% Convert the videos to sequences using the CN as a feature extractor.
% The data is first resized to match the input size of the network
inputSize = netCNN.Layers(1).InputSize(1:2);
layerName = "pool5-7x7_s1";
tempFile = fullfile(tempdir,"hmdb51_org.mat");
if exist(tempFile,'file')
   load(tempFile,"sequences")
else
   numFiles = numel(files);
   sequences = cell(numFiles,1);

   for i = 1:numFiles
       fprintf("Reading file %d of %d...\n", i, numFiles)

       video = readVideo(files(i));
       video = centerCrop(video,inputSize);

       sequences{i,1} =
activations(netCNN,video,layerName,'OutputAs','columns');
```

```
    end

    save(tempFile,"sequences","-v7.3");
end
%% Prepare Training Data
% Partition data into training and validation data
numObservations = numel(sequences);
idx = randperm(numObservations);
N = floor(0.85 * numObservations);
idxTrain = idx(1:N);
sequencesTrain = sequences(idxTrain);
labelsTrain = labels(idxTrain);
idxValidation = idx(N+1:end);
sequencesValidation = sequences(idxValidation);
labelsValidation = labels(idxValidation);
%% Create LSTM Network
% Network layers:
% Sequence input layer matching feature dimension of feature vectors
% BiLSTM layer with 2000 hidden units and dropout layer
% Fully connected layer with an output size corresponding to the number of
classes
% Softmax layer
% Classification layer.
numFeatures = size(sequencesTrain{1},1);
numClasses = numel(categories(labelsTrain));
layers = [
    sequenceInputLayer(numFeatures,'Name','sequence')
    bilstmLayer(2000,'OutputMode','last','Name','bilstm')
    dropoutLayer(0.5,'Name','drop')
    fullyConnectedLayer(numClasses,'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','classification')];
%% Specify Training Options
miniBatchSize = 16;
numObservations = numel(sequencesTrain);
numIterationsPerEpoch = floor(numObservations / miniBatchSize);
options = trainingOptions('adam', ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',1e-4, ...
    'GradientThreshold',2, ...
    'MaxEpochs',3, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{sequencesValidation,labelsValidation}, ...
    'ValidationFrequency',numIterationsPerEpoch, ...
    'Plots','training-progress', ...
    'Verbose',false);
%% Train LSTM Network
[netLSTM,info] = trainNetwork(sequencesTrain,labelsTrain,layers,options);
%% Calculate the classification accuracy of the network on the validation set.
```

```
YPred = classify(netLSTM,sequencesValidation,'MiniBatchSize',miniBatchSize);
YValidation = labelsValidation;
accuracy = mean(YPred == YValidation)
%% Assemble Video Classification Network
% Transform the videos into vector sequences using CN, then classify using
% layers from LSTM network.
% Add Convolutional Layers
cnnLayers = layerGraph(netCNN);
% Remove the input layer and the layers after the pooling layer used
% for the activations
layerNames = ["data" "pool5-drop_7x7_s1" "loss3-classifier" "prob" "output"];
cnnLayers = removeLayers(cnnLayers,layerNames);
% Add Sequence Input Layer
inputSize = netCNN.Layers(1).InputSize(1:2);
averageImage = netCNN.Layers(1).Mean;
inputLayer = sequenceInputLayer([inputSize 3], ...
   'Normalization','zerocenter', ...
   'Mean',averageImage, ...
   'Name','input');
% Add the sequence input layer to the layer graph.
layers = [
   inputLayer
   sequenceFoldingLayer('Name','fold')];
lgraph = addLayers(cnnLayers,layers);
lgraph = connectLayers(lgraph,"fold/out","conv1-7x7_s2");
% Add LSTM Layers
lstmLayers = netLSTM.Layers;
lstmLayers(1) = [];
layers = [
   sequenceUnfoldingLayer('Name','unfold')
   flattenLayer('Name','flatten')
   lstmLayers];
lgraph = addLayers(lgraph,layers);
lgraph = connectLayers(lgraph,"pool5-7x7_s1","unfold/in");
lgraph = connectLayers(lgraph,"fold/miniBatchSize","unfold/miniBatchSize");
analyzeNetwork(lgraph)
%% Assemble the network
net = assembleNetwork(lgraph)
%% Classify Test Data
predictions = strings([100,3]);
testClasses = ["car" "deaf" "finish" "friend" "future" "go-out" "must" "now"
"who" "why"];
for class = 1:length(testClasses)
   classFolder = testFolder + "\" + testClasses(class);
   testFiles = dir(classFolder + "\*.mp4");
   for fileIndex = 1:length(testFiles)
       prediction = strings([1,3]);
       fileName = testFiles(fileIndex).folder + "\" +
testFiles(fileIndex).name;
```

```
        video = readVideo(fileName);
        video = centerCrop(video,inputSize);
        x = {video};
        YPred = classify(net,{video});
        trueClass = split(testFiles(fileIndex).folder,'\');
        trueClass = categorical(trueClass(end));

        prediction(1,1) = fileName;
        prediction(1,2) = trueClass;
        prediction(1,3) = YPred;
        row = 10*(class-1)+fileIndex;
        predictions(row,:) = prediction;
    end
end
%%
%% Helper Functions
% read in video file and return |H|-by-|W|-by-|C-|by-|S| array, where
% |H|, |W|, |C|, and |S| are the height, width, number of channels,
% and number of frames of the video, respectively.
function video = readVideo(filename)
vr = VideoReader(filename);
H = vr.Height;
W = vr.Width;
C = 3;
% Preallocate video array
numFrames = floor(vr.Duration * vr.FrameRate);
video = zeros(H,W,C,numFrames);
% Read frames
i = 0;
while hasFrame(vr)
    i = i + 1;
    video(:,:,:,i) = readFrame(vr);
end
% Remove unallocated frames
if size(video,4) > i
    video(:,:,:,i+1:end) = [];
end
end
%%
% Crop the longest edges of a video and resize to input size
function videoResized = centerCrop(video,inputSize)
sz = size(video);
if sz(1) < sz(2)
    % Video is landscape
    idx = floor((sz(2) - sz(1))/2);
    video(:,1:(idx-1),:,:) = [];
    video(:,(sz(1)+1):end,:,:) = [];

elseif sz(2) < sz(1)
```

```
    % Video is portrait
    idx = floor((sz(1) - sz(2))/2);
    video(1:(idx-1),:,:,:) = [];
    video((sz(2)+1):end,:,:,:) = [];
end
videoResized = imresize(video,inputSize(1:2));
end
%%
% Return a list of files and labels from dataset in dataFolder
function [files, labels] = hmdb51Files(dataFolder)
fileExtension = ".mp4";
listing = dir(fullfile(dataFolder, "*", "*" + fileExtension));
numObservations = numel(listing);
numSubsetObservations = floor(numObservations);
if nargin == 2
    idx = randperm(numObservations,numSubsetObservations);
    listing = listing(idx);
end
files = strings(numSubsetObservations,1);
labels = cell(numSubsetObservations,1);
for i = 1:numSubsetObservations
    name = listing(i).name;
    folder = listing(i).folder;

    [~,labels{i}] = fileparts(folder);
    files(i) = fullfile(folder,name);
end
labels = categorical(labels);
end
```

Appendix B: Code for Python Classifier

```
import os
import cv2
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from moviepy.editor import *
import pandas as pd

from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model

seed_constant = 23
np.random.seed(seed_constant)
random.seed(seed_constant)
tf.random.set_seed(seed_constant)

# Get Names of all classes
classes_list = os.listdir('ASLLRP\\train')

image_height, image_width = 64, 64
max_images_per_class = 200
train_dir = f'D:\\Uni\\capstone\\ASLLRP\\train'
test_dir = f'D:\\Uni\\capstone\\ASLLRP\\test'
model_output_size = len(classes_list)

def frames_extraction(video_path):
    # take in video and output array of normalised frames
    frames_list = []
    video_reader = cv2.VideoCapture(video_path)

    while True:
        success, frame = video_reader.read()
```

```
        if not success:
            break

        resized_frame = cv2.resize(frame, (image_height, image_width))
        normalized_frame = resized_frame / 255
        frames_list.append(normalized_frame)

    video_reader.release()
    return frames_list


def create_dataset():
    # extract videos in each class and return features and labels of each
video
    temp_features = []
    features = []
    labels = []

    for class_index, class_name in enumerate(classes_list):
        print(f'Extracting Data of Class: {class_name}')
        files_list = os.listdir(os.path.join(train_dir, class_name))

        for file_name in files_list:
            video_file_path = os.path.join(train_dir, class_name,
file_name)
            frames = frames_extraction(video_file_path)
            temp_features.extend(frames)

        features.extend(random.sample(temp_features, 200))
        labels.extend([class_index] * max_images_per_class)
        temp_features.clear()

    features = np.asarray(features)
    labels = np.array(labels)

    return features, labels

features, labels = create_dataset()
one_hot_encoded_labels = to_categorical(labels)
```

```
features_train, features_test, labels_train, labels_test =
train_test_split(features, one_hot_encoded_labels, test_size = 0.15,
shuffle = True, random_state = seed_constant)

def create_model():
    # construct convolutional neural network architecture
    model = Sequential()

    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation =
'relu', input_shape = (image_height, image_width, 3)))
    model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation =
'relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2, 2)))
    model.add(GlobalAveragePooling2D())
    model.add(Dense(256, activation = 'relu'))
    model.add(BatchNormalization())
    model.add(Dense(model_output_size, activation = 'softmax'))

    model.summary()

    return model

model = create_model()

print("Model Created Successfully!")

plot_model(model, to_file = 'model_structure_plot.png', show_shapes =
True, show_layer_names = True)

# Stop training early if the model's validation loss does not decrease
after 15 consecutive epochs
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience =
15, mode = 'min', restore_best_weights = True)

model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam',
metrics = ["accuracy"])

# Start Training
```

```
model_training_history = model.fit(x = features_train, y = labels_train,
epochs = 50, batch_size = 4 , shuffle = True, validation_split = 0.2,
callbacks = [early_stopping_callback])
model_evaluation_history = model.evaluate(features_test, labels_test)

# Save model
date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt,
date_time_format)
model_evaluation_loss, model_evaluation_accuracy =
model_evaluation_history
model_name =
f'Model___Date_Time_{current_date_time_string}___Loss_{model_evaluation_lo
ss}___Accuracy_{model_evaluation_accuracy}.h5'
model.save(model_name)

# # Load model (Above code can be commented out and replaced with below
code if a model already exists)

# model = create_model()
# plot_model(model, to_file = 'model_structure_plot.png', show_shapes =
True, show_layer_names = True)
#
model.load_weights('Model___Date_Time_2022_10_23__20_21_40___Loss_0.783248
7225532532___Accuracy_0.7333333492279053.h5')


def make_average_predictions(video_file_path):

    video_reader = cv2.VideoCapture(video_file_path)
    window_size = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

    predicted_labels_probabilities_np = np.zeros((window_size,
model_output_size), dtype = np.float)

    for frame_counter in range(window_size):

        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter)
        _ , frame = video_reader.read()
```

```
        resized_frame = cv2.resize(frame, (image_height, image_width))
        normalized_frame = resized_frame / 255


        predicted_labels_probabilities =
model.predict(np.expand_dims(normalized_frame, axis = 0))[0]
        predicted_labels_probabilities_np[frame_counter] =
predicted_labels_probabilities

    predicted_labels_probabilities_averaged =
predicted_labels_probabilities_np.mean(axis = 0)
    predicted_labels_probabilities_averaged_sorted_indexes =
np.argsort(predicted_labels_probabilities_averaged)[::-1]

    label = predicted_labels_probabilities_averaged_sorted_indexes[0]
    predicted_class_name = classes_list[label]
    predicted_label_probability =
predicted_labels_probabilities_averaged[label]

    video_reader.release()
    return predicted_class_name, predicted_label_probability


df = pd.DataFrame(columns=["File", "True Class", "Predicted Class",
"Predicted Class Probability"])
for root, _, files in os.walk(test_dir, topdown=False):
    for name in files:
        input_video_file_path = os.path.join(root, name)
        print("\nPredicting file: " + input_video_file_path)
        predicted_class, predicted_class_probability =
make_average_predictions(input_video_file_path)
        trueClassName = root.split('\\')[-1]
        df.loc[len(df)] = [input_video_file_path, trueClassName,
predicted_class, round(predicted_class_probability, 2)]

df.to_csv('Predictions.csv', index=False)
```

Appendix C: Project Communication Log

Project Title:		Automatic Sign Language Recognition Project		
Student Name:		Thomas Gallagher	**Supervisor Name:**	Thuy Pham
Date	**Event**	**Topic of Communication**	**Outcome**	
06/02/22	Email	Confirmation of project	My supervisor confirmed the project I would be working on	
07/02/22	Email	Project resources	My supervisor supplied me with some resources that I could utilise for the project.	
22/02/22	Email	Project questions	My supervisor wanted to confirm that I had read through the resources and if I had any questions.	
22/02/22	Email	Project questions and resources	I confirmed I had no questions but couldn't access the resources my supervisor had sent anymore.	
22/02/22	Email	Project resources	My supervisor resupplied me with the resources.	
08/04/22	Email	Forms	I detailed the ethics form and safety plan to my supervisor for approval.	
08/04/22	Email	Progress update	My supervisor asked how my progress was going.	
09/04/22	Email	Forms	My supervisor approved both my supplied forms.	
09/04/22	Email	Progress update	I updated my supervisor on my progress, which was waiting for approval on the BOBSL dataset and attempting to get the code to run on my new PC as my laptop had stopped working.	
11/05/22	Email	Progress update	My supervisor asked how my progress was going.	
13/05/22	Email	Progress update	I described my struggles with running the BSL-1K code and PC issues.	
23/05/22	Email	Uploading data	My supervisor asked me to upload the data and progress I had achieved so far	

			to a UTS shared folder.
25/05/22	Email	Uploading data and progress update	I uploaded my existing work to the shared folder and notified my supervisor, also detailing my progress and current stage of the project.
26/05/22	Email	Re-evaluation of work priority	My supervisor recommended that I could always write my own approach and implementation rather than attempting to utilise the BSL-1K code. She also asked for details on downloading the BOBSL dataset,
31/05/22	Email	Downloading BOBSL dataset instructions	I sent my supervisor links and guidance on how I downloaded and was testing the BSL-1K code which includes the BOBSL dataset.
15/08/22	Email	Confirmation of accessibility and progress update	My supervisor asked about my accessibility to the BOBSL dataset and my progress on the project.
15/08/22	Email	Progress update and GitHub addition	I showed my supervisor my email sent to access the BOBSL dataset and the responding email. I also let my supervisor know that I wasn't sure if I should work on my own model or improve the BSL-1K work. Also created and shared a GitHub repository of my work. Attempted to schedule a meeting on Monday at 5:30.
15/08/22	Email	Progress update and new member collaboration	My supervisor suggested working on the existing BSL-1K if I was struggling with model creation. She also suggested working with another student who was working on the same capstone project and offered to introduce me to him. My supervisor wasn't available to meet on Monday, so suggested scheduling another meeting in 2 weeks if I was still struggling.
15/08/22	Email	Collaboration	I advised my supervisor that I would be happy to discuss the project with the new team member.
15/08/22	Email	Collaboration	My supervisor introduced me to the student working on the same project.

22/08/22	Email	UTS HPC accessibility	My supervisor sent a link to UTS High Performance Computing.