# COMPUTER VISION FOR LIP READING

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2024

**Jack Pay**

Supervised by Dr Terence Morley

Department of Computer Science

# Contents

**Word Count: 14,272**

# List of Tables

# List of Figures

# Abbreviations

**AI** Artificial Intelligence. 22

**ANN** Artificial Neural Network. 10–12, 24, 29, 32–35, 37, 39, 45, 60, 61, 63–66, 91

**ANNs** Artificial Neural Networks. 28, 32, 34

**ASR** Automatic Speech Recognition. 32, 42

**Bi-GRU** Bidirectional Gated Recurrent Unit. 35

**Bi-LSTM** Bidirectional Long Short Term Memory. 14, 22, 35, 70, 74, 75, 90, 91

**CER** Character Error Rate. 32

**CNN** Convolutional Neural Network. 6, 14, 23, 35, 37, 38, 44, 70, 71, 74, 85, 90

**CNNs** Convolutional Neural Networks. 22, 35

**CSF** Computational Shared Facility. 6, 45–49

**CTC** Connectionist Temporal Classification. 6, 14, 23, 42, 43, 70, 87–92

**CV** Computer Vision. 90

**DHL** Disabling Hearing Loss. 20–22, 93

**DL** Deep Learning. 35

**GPU** Graphical Processing Unit. 28, 46, 47

**GRU** Gated Recurrent Unit. 35, 44, 85

**GUI** Graphical User Interface. 7, 15, 23, 24, 31, 45, 64–67, 69, 91, 92

**LR** Learning Rate. 6, 15, 39–42, 62, 63, 68, 70, 72, 74, 76–82, 84, 85, 88, 90

**LSTM** Long Short Term Memory. 6, 14, 22, 23, 35, 36, 44, 70, 71, 74

**ML** Machine Learning. 6, 7, 11, 21, 23, 28, 29, 32–34, 39, 46, 51, 60, 61, 90, 91, 93

**NLP** Natural Language Processing. 10, 12, 27, 31

**P2V** Phoneme to Viseme. 7, 31, 58, 59

**RNN** Recurrent Neural Network. 35

**SGD** Stochastic Gradient Descent. 34, 39, 41

**SRS** Simple Random Sampling. 60

**SRU** Simple Recurrent Unit. 6, 36

**UoM** University of Manchester. 45, 47

**WER** Word Error Rate. 5, 14, 32, 44, 61, 72, 74, 76, 78, 81, 82, 84, 85, 88, 90, 91, 102

**WHO** World Health Organisation. 20

# Glossary

**activation function** An activation function is a mathematical function used to decide the output of a neuron within an Artificial Neural Network (ANN). 6, 32–35

**corpora** Plural of corpus. A text corpus is a text dataset. 31

**data augmentation** Different techniques for enhancing and increasing the size of a dataset [61]. There are various methods such as geometric transformations, cropping and data flipping. 72–75, 82, 83

**dropout** A method to reduce overfitting [57]. Randomly, inputs to nodes are set to 0, excluding certain nodes from training runs. Keras gives an implementation of Dropout layers[1]. 71, 75

**feature extraction** Feature extraction refers to the process of identifying and extracting intrinsic features from data for various other processes, from classification to clustering [46]. There are different methods for feature extraction, varying based on the task required. Feature extraction is the first step among many computer vision and NLP processes, putting data into a format so that it can be easily processed. 29, 38, 57

**fine-tuning** A method of transfer learning where an already trained, or pre-trained, model is trained a second time on a different task. The benefit being the transferal of knowledge from the pre-trained model to a new model for a new task. 15, 23, 66–69

**GLips** GLips is a dataset for German lip reading [59]. It is a collection of 250,000 videos from speakers in the Hessian Parliament. 50

[1] https://keras.io/api/layers/regularization_layers/dropout/

**LRS2** Lip Reading Sentences in the Wild (LRS2) is a large-scale dataset for English lip reading [63]. It was collected as a follow-up to the popular LRW and intended to instead understand full sentences. 50, 52

**LRS3-TED** LRS3-TED is one of the largest datasets for English lip reading [3]. It was collected from a set of TED talks to avoid restrictions associated with the data and to better compare different lip reading systems. 50

**LRW** Lip Reading in the Wild (LRW) is the first large-scale dataset for English lip reading [12]. It is an annotated dataset including 500 English words, each being uttered 1000 times. 6, 11, 14, 21, 44, 50, 52–55, 57, 58, 72, 73, 90, 92

**LRW-1000** LRW-1000 is a dataset for Chinese Mandarin lip reading [77]. It is a collection of 718,018 videos. 50

**multi-class classification** Classification of data samples into multiple classes. Can be single-label or multi-label. 34, 57, 58

**multi-label classification** Classification of data samples into a set of multiple classes. Single samples can be classified into possibly several classes. 58, 87

**one hot encoding** A technique for encoding class labels in ML, involving assigning N-dimensional vectors to samples, where N represents the number of classes. Within these vectors, a 0 is stored at position $i$ if the sample does not belong to the class $i$, and 1 if it does. 52, 71, 88

**overfitting** Overfitting refers to the problem of an ANN not generalising to data. Models that have overfit perform well on the training data but not on external, unseen data [78]. 14, 39, 52, 60, 62, 70–76, 78, 82–84

**phoneme** The fundamental and distinct sounds that are made during speech, used to distinguish between different words. 6, 9, 14, 21, 25, 27, 29, 31, 34, 50, 57–59, 70, 87–89, 91

**single-label classification** Classification of data samples into a set of multiple classes. Single samples can be classified into only one class. 57, 87

**speech disfluencies** Speech that doesn't flow well and may contain repeated words, self-corrections and filled pauses [37]. 20, 35, 38

**Transformer** The Transformer architecture is another ANN architecture. It is made unique by its use of multi-head attention blocks, positional encoding and *Add&Norm* layers [68]. The Transformer architecture has been used to great success for various visual [27, 17, 38] and NLP applications. 14, 22, 23, 35, 44, 70, 83, 84, 87, 90, 92

**underfitting** Underfitting refers to the problem of an ANN not generalising to data at all. The model is unable to capture the structure of the training, testing or validation datasets. 72, 73

**viseme** The visual counterparts to phonemes; the speech sounds that form similar lip shapes. 6, 9, 14, 22, 25–28, 31, 34, 50, 58, 59, 70, 87–89, 91

# Listings

# Abstract

Lip reading is an often ignored but crucial aspect of communication. It is essential for deaf people but is also used to a degree by those without hearing impairments. This project aims to research different methods for automating lip reading using Machine Learning.

We utilised LRW, a popular lipreading dataset provided by the BBC. With this, we generated useful data using MediaPipe and Python, and conducted a set of experiments to compare different model architectures trained for lip reading.

We assessed the performance of basic Convolutional Neural Network (CNN), Long Short Term Memory (LSTM) and Bidirectional Long Short Term Memory (Bi-LSTM) architectures, tuning different hyperparameters in an attempt to make the best model possible. We trained and evaluated Transformer architectures and utilised cross-attention to compare features from two separate modalities: both landmarks and image data. In an attempt to reduce the effects of overfitting, we also assessed data augmentation and dropout.

In the final stages of experimentation, we also trialled different loss metrics, primarily Connectionist Temporal Classification (CTC) loss, and changed the classes utilised for training. As opposed to the words being uttered, we tested the performance when videos are classified to the letters, phonemes and visemes uttered.

Overall, seventeen experiments were conducted. A summary of the best models developed for lip reading is displayed in Table 5.1, displaying up to 97% accuracy, less than 10% loss, and a Word Error Rate (WER) of just 13.7%. A summary of all of the results of these experiments is shown in Table A.1. A Bi-LSTM model, that utilised lip landmarks as input, proved to be the best architecture. We also discovered that utilising visemes, rather than letters or phonemes produced the best loss and WER for lip reading.

Finally, once we had assessed enough different architectures, a Graphical User Interface (GUI) was then developed. We designed the application to have two modes: inference and fine-tuning. Inference mode was provided to run pretrained models in real-time or input videos. Fine-tuning allowed recording of new data and fine-tuning existing models that could then be saved or loaded in as necessary. The GUI was developed to enable thorough customisation over inference, such as a prediction threshold, and configuration of fine-tuning, for example the number of epochs and Learning Rate (LR).

# Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

# Acknowledgements

Firstly, I would like to thank my supervisor, Dr Terence Morley, without whom I would not have been able to complete this project. Thank you for all of your guidance and support throughout this project.

I want to also extend my gratitude to the University of Manchester and the Department of Computer Science. The past three years have been challenging, thrilling and unforgettable. I have learnt so much. Thank you.

I want to thank the BBC for their generous provision of valuable data that was key to developing this research.

Finally, thank you to my family for their continued support throughout my time at University. Thank you for your wise advice, a place to stay whenever it was needed and for always being there. I couldn't have done this without you.

# Data Declaration

For non-commercial individual research and private study use only. BBC content included courtesy of the BBC

# Chapter 1

# Introduction

## 1.1 Motivation

Deafness is estimated to be the fourth leading disability globally [13]. In 2018, the World Health Organisation reported that 466 million people (over 6.1% of the world's population) were affected by Disabling Hearing Loss (DHL) [71]. This is estimated to reach 700 million people by 2050 [72]. If accurate, one in ten people will be affected by deafness.

This trend would mean that treatments for DHL, such as cochlear implants and hearing aids, will rise in demand in the coming years. Accessibility features in videos, like closed captions and sign language, will become more critical. Currently, these technologies are powered mainly by processing audio clips and synchronising them in real-time to the correct place within videos. However, this is harder to do in real-time and is made even more difficult by a range of factors. Accent, speech disfluencies and background noise can hugely impact the transcription of words being said.

A major factor that impacts speech processing is the shape of the moving mouth, even in those without hearing loss [70]. Incorporating visual components, in addition to audio, provides more precise speech processing results [9].

Utilising visual components could improve speech recognition technology. For noise, being able to select a single talker and focus on their speech could help an individual with hearing loss to understand them. Modern-day speech recognition would become confused within this noisy environment (unless uni-directional microphones were used), possibly trying to listen to many speakers at once. This technology wouldn't just aid those with DHL but also individuals with auditory processing disorders, such as Autism.

Currently, no software supports this functionality and there is little existing software able to overcome the difficulties present within speech.
Most people with DHL live in low- and middle-income countries [72] and therefore have poorer access to medical treatment for their condition. Education within these regions may also be lacking, leading to individuals not learning the necessary skills to understand those around them [58, 49]. Without treatment, for hearing loss, deafness can become debilitating.
Therefore, there is a high, and rising, demand for software able to use visual cues for lip reading. With an increasing proportion of the population suffering from DHL, treatment and accessibility are becoming far more important. Visual lip reading could be used to improve current speech recognition software, aid those with hearing and auditory processing conditions, and educate people first learning how to read lips.

## 1.2 Machine Learning for Lip Reading

Machine Learning (ML) is now used for all manner of applications to great success. From stock market prediction to image recognition and even generating artwork, ML can carry out all manner of complex jobs.
Lip reading using ML is cheap, fast and can be improved over time. ML models possess the complexity to capture essential nuances within data. Lightweight models can be produced that could run on mobile devices. This makes ML perfect for the role of lip reading. Lip reading is also largely temporal in nature, as shown in Figure 1.1, a property that ML greatly suits. The only downside of ML is the vast amount of data required, countered only by the abundance of lip reading data and ease of collecting more. In the modern day, there are millions of videos of people speaking, available in just a few clicks online. The average person utters over a hundred words per minute making it easy to collect new data if necessary.
The purpose of this paper is to train numerous different models, in tandem, to find

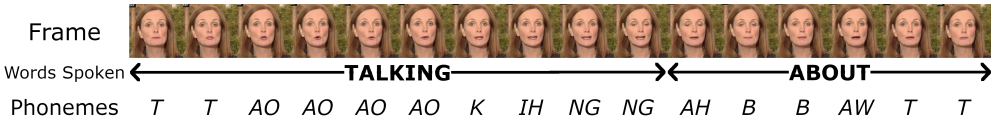



Figure 1.1: A lip reading example from LRW [12]. This shows the temporal nature of speech, with words spread across frames and being comprised of phonemes. In the example, two words are said: “TALKING” and “ABOUT”.

the best capable of recognising spoken language. We aim to explore and expand upon other findings within this field and create our own lip reading models.
Leading model architectures such as Bidirectional Long Short Term Memory (Bi-LSTM) Neural Networks, Long Short Term Memory (LSTM) Neural Networks, Convolutional Neural Networks (CNNs) and Transformers will be investigated and compared. The performance of such architectures is outstanding for various applications such as video classification, translation and text summarisation. This makes them promising candidates for lip reading.
However, lip reading is no easy feat. Various factors make it extremely difficult to predict spoken words using only visual features. The rate of speech, homophones, partial visibility, local dialects, the size of the vocabularies and hard-to-distinguish visemes can make lip reading difficult for humans, let alone computers.
Context can be used to give models a way to distinguish between different word senses and homophones, however, this has its own difficulties. Dealing with context can lead to large models and complexity when designing model architectures.

## 1.3 Applications

The applications of lip reading are expansive. Already, there are modern applications, such as HeyGen[1], which do audio and visual multi-language translation, including altering lip movements to seem more realistic. Imagine being in a meeting with someone from across the world, both able to speak in your native languages, whilst AI handles the translation.
Another application is accessibility support. Cochlear implants and hearing aids are employed by those with hearing loss but they are expensive and invasive, possibly requiring surgery. Lip reading technology could be fitted into a pair of glasses to help people with DHL on the go. What about even combining the two options above? Imagine a pair of contact lenses or headphones akin to Douglas Adams' Babel fish [2].
Lastly, lip reading could be crucial for mediums that are visual only. The world is filled with outdated, visual-only surveillance cameras. By employing lip reading technology, words spoken by criminals could be reconstructed and used as evidence for law enforcement. Sound in videos can become corrupted and unavailable but with lip reading, important meeting notes could be reconstructed so that information is never

[1] https://www.heygen.com/

lost. Even noise or distance in environments could be circumvented, allowing two people to communicate whilst across a busy hall without a phone call. The applications are numerous and thrilling.

## 1.4 Aims and Objectives

The primary aim of this research is to create a set of ML models capable of lip reading to find the best architectures. To achieve this there will be various objectives:

1. Find and gain access to annotated datasets that contain videos of participants speaking
2. Detect people's faces and localise to their lips, giving image crops or lip landmarks
3. Develop a data pipeline able to generate useful features for training
4. Produce a set of models able to capture and identify words being spoken using only visual features
5. Optimise the various models to achieve a high performance
6. Compare different architectures and lip reading methods including:
    - CNN architecture followed by LSTM units
    - A bi-directional architecture
    - Lip reading based on visual features, lip features and both at once
    - An attention-based and Transformer architecture
    - Using CTC loss to train lip reading
7. Develop a basic Graphical User Interface (GUI) to assess model performance in real-time
8. Extend the GUI to allow for model fine-tuning

## 1.5 Report Structure

This report is split into 5 chapters:

- Chapter 2 gives a thorough treatment of the background knowledge for the subject area, expanding on some of the physical complications and aspects of lip reading It explains the theory behind various Artificial Neural Network (ANN) architectures used in the later chapters
- Chapter 3 addresses the design of the various project stages, including Python software tools that were utilised and the experiment design. It explains the structure of the Graphical User Interface, giving some examples of it in use
- Chapter 4 evaluates the different models and experiments, analysing the metrics captured and giving interpretations
- Chapter 5 summarises the work completed and offers some insights on where this work can be taken next

# Chapter 2

# Background

## 2.1 Lip Reading

Lip reading is a crucial means of communication. It is one of the few mediums of communication available for people who are hard of hearing or deaf, but all people employ lip reading to a degree. The ability to process the shape of the mouth and lips is essential to everyone during speech processing [70].

Lip reading, or speechreading, refers to being able to understand what a person is saying by using only the visual movements of the face. It is a multimodal communication method; lip reading requires information including the shape of the lips, movement of the tongue and position of the teeth [18]. Context is also crucial for lip reading, having been shown to have a huge impact on the ability to recognise different words [64].

In communication, there are various methods to break down and categorise spoken language. The most popular would likely be phonemes: the fundamental and distinct sounds that are made during speech. Visemes are another fundamental unit of language. Visemes represent the visual counterparts to phonemes. They are the speech sounds that form similar lip shapes and are thus paramount for lip reading.

However, here is our first problem with lip reading: we do not have a set of consistent visemes. This is primarily an issue highlighted within the domain of animation; animators have to create realistic facial expressions depending on the words being spoken. In the early days of animation, animators realised that different phonemes shared the same facial expression and, therefore, reduced the number of different faces they had to animate. One of the most notable viseme sets was Disney's, as shown in Figure 2.1, which categorises mouth positions into just twelve parts. There are typically ten to fourteen different visemes but there is no global viseme set. As shown in Figure 2.2,

Figure 2.1: The twelve classic Disney visemes. These show the different lip shapes, used by Disney for the 1937 film, Snow White and the Seven Dwarfs, to express different voice noises. Image source: https://docs.cryengine.com/display/SDKDOC2/Phonemes+and+Visemes


Figure 2.2: OneClick's DAZ Gen8.1. viseme example. These show a more modern, three-dimensional representation of the different visemes used for animation. Image source: https://crazyminnowstudio.com/docs/salsa-lip-sync/modules/recommendations/

different animators had different sets of visemes.

There is another glaring issue with visemes and so lip reading, one that is similar to the everyday homophone. The various viseme sets almost always have fewer categories than phonemes do. The consensus in English is that there are 44 unique phonetic sounds, presented in Figure 2.3. This number is far larger than any set of visemes, which highlights the second major issue with lip reading: the correspondence between phonemes and visemes is tenuous. This is because of visual homophones or homophenes: words that when spoken have different audio components but the same visual components. One of the most famous, and humorous, examples of this is the phrase “I love you” and “elephant juice”. When spoken aloud these phrases are as different as can be but, for a lip reader, context is needed to tell them apart. Therefore, lip reading is made more complex by the difficult synchronisation between what is spoken and what is visualised.

Some studies, such as Bear’s “Which phoneme-to-viseme maps best improve visual-only computer lip-reading?”, have examined which viseme sets best map to the phonemes [6]. However, this study only analyses English phonemes and visemes. As of the time of conducting this research there are 7000 languages spoken worldwide [69] which might require their own set of phonemes and visemes. There do exist metaphonemes, which are cross-language phonemes [8]. However, this would require further analysis into the mapping between these and visemes.

Lip reading suffers from all of the difficulties present within classical Natural Language Processing, such as homophones, the context being lost, variation in language and ambiguity.

Overall, lip reading is an immensely challenging task, one which many fluent speakers

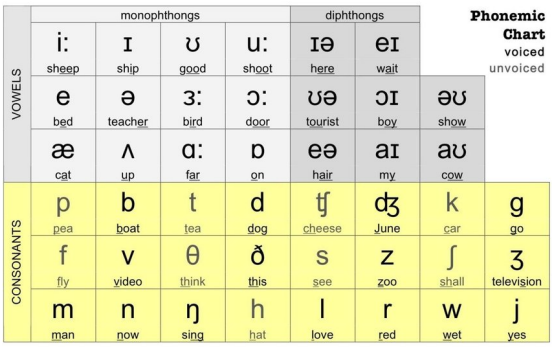



Figure 2.3: The 44 Phonemes of the English Language [19]. This represents the widely accepted set of English phonemes, as first presented by Underhil in 2008.

can struggle with. From a lack of understanding surrounding visemes, to the thousands of languages spoken around the world, lip reading is hard for people, let alone computers.

## 2.2 Technologies

This section will delve into the different technologies involved in this research. Various decisions were made surrounding the different tools to employ for data generation, producing a working lip reading model and designing a GUI to showcase the lip reading model.

### 2.2.1 Keras, TensorFlow and PyTorch

To train ML models there are two primary libraries: TensorFlow[1] and PyTorch[2]. These provide various functionality, such as being able to define Artificial Neural Networks, perform mathematical operations on batched data and format sparse or big data. It is important to note that TensorFlow contains another API called Keras[3].
There is wide discourse as to which tool is the best. PyTorch is low level, high-performance, easier to debug but less readable. TensorFlow is high and low-level, high-performance, has better readability but is harder to debug. Keras is the most popular of all three. Whilst PyTorch is written in Lua and TensorFlow in a combination of C++, CUDA and Python, Keras is only written in Python. Keras has lower performance but makes up for this with its simplicity and code readability [67]. Typically, TensorFlow is regarded as being used more in industry whilst PyTorch is used primarily for research.
I decided to go with the popular consensus and use Keras. Its simplicity and readability would be advantageous since I have not made use of any of these tools before. Furthermore, as I am more experienced in Python, this would better suit my skill set. Although it is lower performance than the other libraries, I will only be creating relatively small models and have access to quite powerful GPUs for training. Furthermore, this research may be useful to get more experience in the industry standard libraries employed for ML.

[1] https://www.tensorflow.org/
[2] https://pytorch.org/
[3] https://keras.io/

### 2.2.2 MediaPipe and Dlib

MediaPipe[4] is an innovative Machine Learning framework developed by Google. MediaPipe provides various functionalities such as gesture tracking, object classification, audio processing and natural language processing. The framework allows for efficient resource management, allowing both GPU and CPU utilisation to increase the performance of running ML models [39]. Shown in Figure 2.4, one notable tool is MediaPipe's Face Landmark Detection[5]. This provides a library for detecting faces within images and estimating hundreds of facial landmarks such as the lips, eyes and nose.

Dlib is a C++ toolkit that contains a diverse selection of Machine Learning solutions. Dlib is primarily a collection of various software components [28], the most notable being its Face Detector[6], shown in Figure 2.5. This class can detect faces within images and return the coordinates of facial landmarks.

Another major technological decision was whether to utilise MediaPipe or Dlib for collecting data to train the lip reading models. A crucial step for this research is being able to localise to a person's lips to collect data for inference, much like a person does in conversation.

In total MediaPipe detects 478 3D facial landmarks, forty of which are directly associated with the lips. Dlib detects sixty-eight 2D facial landmarks, nineteen of which are related to the lips. Further investigation could be done into which performs the best at localising to faces but this difference will likely be insignificant. The more important difference between these two libraries is their landmark data.

A major part of ML is feature extraction. Facial landmarks extracted from images could be useful to train an ANN to focus directly on the shape of the lips when different phonemes are being said. These could significantly cut down on noise within data, using transfer learning to make a more accurate model.

Consequently, the clear choice is to use MediaPipe. It provides more information associated with the lips: there are more than double the landmarks, all of which are a higher dimension than the landmarks generated by Dlib.

[4] https://developers.google.com/mediapipe

[5] https://mediapipe-studio.webapps.google.com/demo/face_landmarker

[6] http://dlib.net/face_detector.py.html

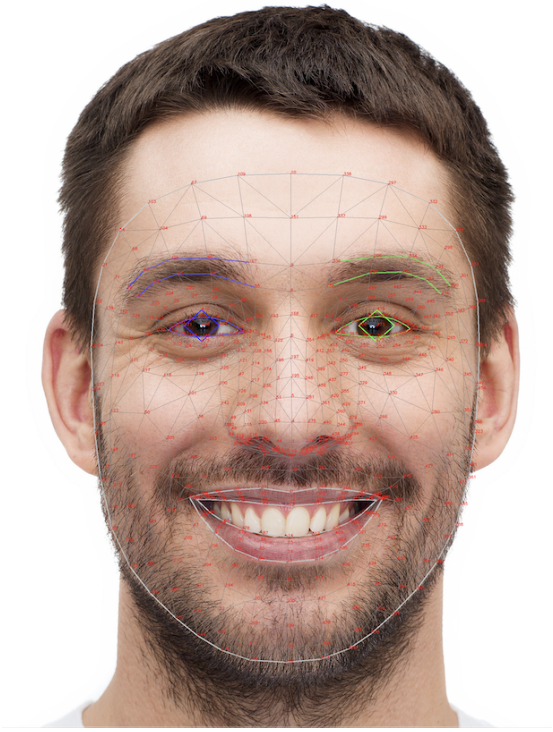


Figure 2.4: Visualizing the 478 MediaPipe facial landmark coordinates. This image gives a map of the different MediaPipe facial landmarks, labelling each with its index. Image source: https://developers.google.com/mediapipe/solutions/vision/face_landmarker

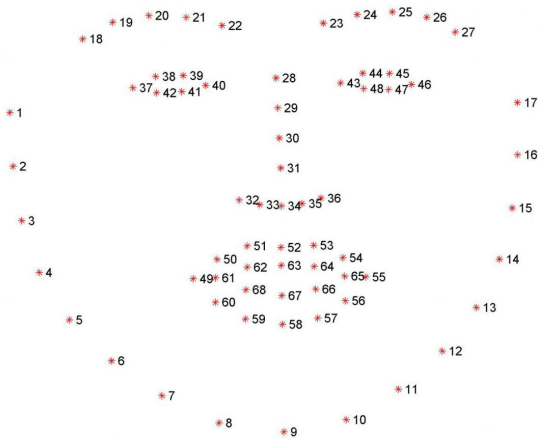



Figure 2.5: Visualizing the sixty-eight Dlib facial landmark coordinates. This gives a map of Dlib's facial landmarks, giving the index of each different landmark. Image source: https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

### 2.2.3 Tkinter

Tkinter[7] is a Python package that interfaces the Tk Graphical User Interface (GUI) toolkit. It is a package for producing various cross-platform GUI applications with a native look and feel [40].
There is much discussion about which Python GUI package to use, some popular choices include PySimpleGUI[8], Flexx[9] and Ipywidgets[10].
For this project, Tkinter will be used due to its simplicity, ability to be run on different Operating Systems and its accessibility (Tkinter is pre-installed on Python) [52]. Tkinter has good flexibility and I have experience with this tool already. Therefore, the focus of the project will remain on lip reading rather than learning a new Python GUI tool.

### 2.2.4 NLTK

NLTK[11] is a Python library for Natural Language Processing and text analytics [50]. It provides various tools for processing text like lemmatisation, tagging, tokenisation and word normalisation. Moreover, NLTK contains numerous corpora.
One useful tool provided by NLTK is the Carnegie Mellon Pronouncing Dictionary (cmudict)[12]. This reduces English words to their constituent phonemes.
As there is no standard set of visemes, NLTK cannot be used as a Phoneme to Viseme (P2V) mapper: a tool that would be crucial for generating viseme data. Research suggests [6] that the best correspondence between phonemes and visemes is Lee [34]. This maps phonemes to six consonant visemes and seven vowels visemes. It also has a silence viseme [34]. This P2V map will be hardcoded into a dictionary to easily convert from phonemes to visemes. So, NLTK could be used to convert words to their constituent phonemes before this P2V mapping is used to convert again to visemes.

[7] https://docs.python.org/3/library/tkinter.html
[8] https://www.pysimplegui.org/en/latest/
[9] https://flexx.readthedocs.io/en/stable/
[10] https://ipywidgets.readthedocs.io/en/stable/
[11] https://www.nltk.org/
[12] https://www.nltk.org/_modules/nltk/corpus/reader/cmudict.html

### 2.2.5 JiWER

JiWER[13] is a Python library that supports various Automatic Speech Recognition (ASR) tasks. JiWER has been used to compare the performance of various ASR systems [55]. It provides various useful functions such as calculating the Word Error Rate (WER) and Character Error Rate (CER), intrinsic for comparing lip reading model performance.
The WER is defined as

$$WER = \frac{S_w + D_w + I_w}{H_w + S_w + D_w} = \frac{S_w + D_w + I_w}{N_w},$$

where $WER$ is the word error rate, $S_w$ the number of substitutions (in the predicted phrase), $D_w$ the number of deletions, $I_w$ the number of insertions, $H_w$ the number of correct words (or hits in the predicted phrase) and $N_w$ is the number of words (in the reference phrase) [55].
The CER is defined similarly, although rather than looking at words it focuses on individual letters

$$CER = \frac{S_c + D_c + I_c}{H_c + S_c + D_c} = \frac{S_c + D_c + I_c}{N_c}.$$

In these equations, the subscript $w$ signifies variables at the word level and subscript $c$ at the character level.

## 2.3 Artificial Neural Networks

An Artificial Neural Network (ANN) is a structure inspired by the brain. It is a set of connected artificial nodes, like mathematical functions, that take input from previous nodes and pass this to later nodes.
This section will cover various fundamental concepts for ML and training an ANN.

### 2.3.1 Softmax

Softmax is a crucial but easy mathematical function, shown in Figure 2.6, often used as the final activation function within ANNs. Softmax is used to normalise a set of numbers so that they sum to 1 [5]. Consequently, it is often used to normalise the output of an ANN to give the probability that data samples belong to each class within

[13] https://pypi.org/project/jiwer/

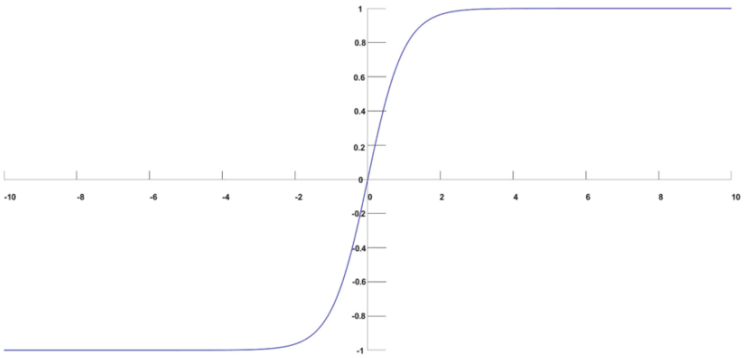


Figure 2.6: Softmax function image [35].

a set.
The Softmax, $\sigma(x)_i$, for each element $x_i$ of a vector is defined as

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}},$$

where there are $K$ classes, $i, j = 1, 2, ...K$ and $\mathbf{x} = (x_1, x_2, ...x_K)$.

### 2.3.2 ReLU

To train an ANN, the outputs of each layer are passed through a nonlinear function: an activation function. Shown in Figure 2.7, ReLU is a commonly used, simple and efficient activation function within ML [32].

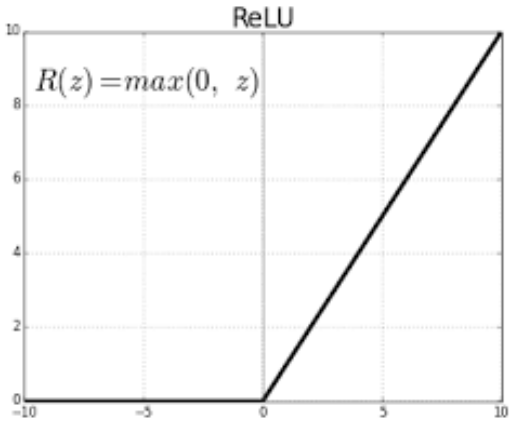



Figure 2.7: Graph of ReLu activation function [24].

ANNs trained using ReLU typically perform better than those trained using other activation functions, such as the tanh function [54].
ReLU, $R$, will output the input, $z$ if it is positive but otherwise give the value 0.

### 2.3.3 Cross-Entropy Loss

Loss is a crucial measurement within ML. Loss measures the difference between the predicted and the true classes of data samples. Most methods for training an ANN, such as Stochastic Gradient Descent (SGD), aim at reducing loss for a set of data samples. But, depending on the task, there are many different loss measurements.
Cross-entropy loss is employed during training of ANNs that output the probability that a given data sample belongs to a class. Cross-entropy measures the difference between the true probability and a model's predicted probability. A value of 0 means that the model perfectly predicts the probability [60].
There are different flavours of cross-entropy, depending on the task. For example, binary cross-entropy is used when there are only two different classes of data. Categorical cross-entropy is used for multi-class classification.
In multi-class classification, an ANN will output the probability that a data sample belongs to each different class. Therefore, categorical cross-entropy is employed to increase this probability for the correct class of a sample and reduce the probability for incorrect classes.
Categorical cross-entropy will be employed during training of lip reading models as this is a multi-class classification problem; we have a video containing one of $X$ words (or phonemes, or visemes).
The formulation for cross-entropy loss, $L$, over $N$ classes is defined as

$$L = -\sum_{i=1}^{N} y_i \log p_i,$$

where $y_i$ is the truth value, of whether a data sample is in class $i$, and $p_i$ is the Softmax probability for the class $i$. For training, there are four classes so this can be expanded further:

$$L = -[y_0 \log p_0 + y_1 \log p_1 + y_2 \log p_2 + y_3 \log p_3].$$

### 2.3.4 Long-Short Term Memory

A Recurrent Neural Network is a type of ANN used in Deep Learning (DL) for sequential data such as text, audio and videos. There are different types of RNN classified by their structure, such as classic RNN, LSTM, GRU, Attention RNN and Transformers. The choice of RNN unit largely depends on the task being carried out, for example the Transformer architecture is primarily used for modern large language models such as ChatGPT[14].

Long Short Term Memory (LSTM) is an RNN cell operation first proposed to overcome the vanishing gradient problem [44]. LSTM overcame many of the shortcomings of early, vanilla RNNs and allowed for both short and longer term dependencies to be established within sequential data.

LSTM is comprised of three gates called the input, output and forget gates. Shown in Figure 2.8, these gates give LSTMs the ability to alter or drop data from a sequence [16]. This makes them useful for processing video data where there are long-term dependencies between frames and speech containing speech disfluencies or features that need to be ignored. The gates are computed using sigmoid activation functions, shown in Figure 2.9. LSTM has also been shown to be successful for video facial recognition tasks in the past [1].

A notable extension to typical RNNs is that of a bidirectional RNN. This allows consideration of both left and right context: events that happened in both the past and future. Bidirectional RNNs have been used to derive Bidirectional Long Short Term Memory (Bi-LSTM) (shown in Figure 2.10) and Bidirectional Gated Recurrent Unit (Bi-GRU). However, the main downside of these networks is that they require training two separate architectures simultaneously.

### 2.3.5 Convolutional Neural Networks

Primarily used to solve image classification tasks, Convolutional Neural Networks (CNNs) are a powerful ANN architecture [48, 15]. CNNs are characterised by three primary components: convolutional layers, max pooling layers and a final set of fully-connected layers [48].

The operation of convolutional layers hinges on filters (also called kernels), comprised of trainable parameters, that are convolved across an image [10, 15]. Convolution can be done on images if the layer is the first within the network, like *Conv-1* in Figure 2.11,

[14] https://chat.openai.com/

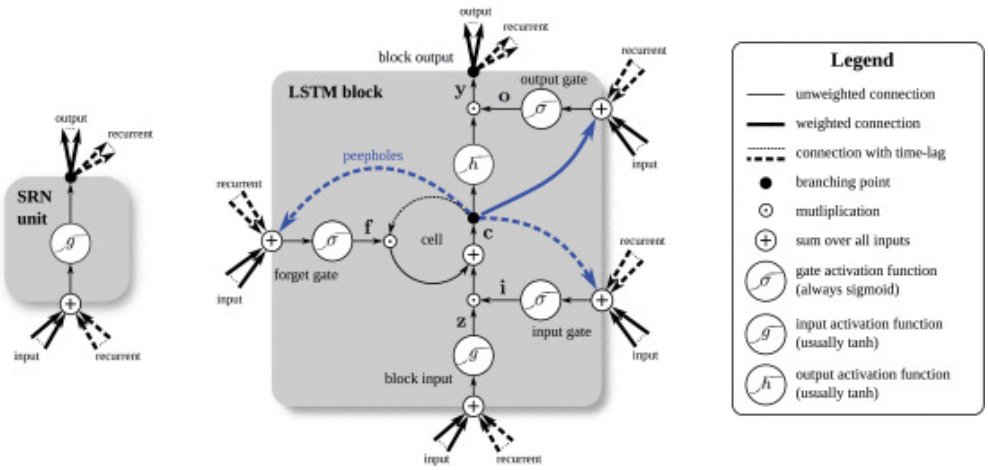



Figure 2.8: Detailed diagrams of a Simple Recurrent Unit (SRU) and Long Short Term Memory (LSTM) [23]. A SRU is shown on the left and an LSTM block on the right as used in the hidden layers of a recurrent neural network.

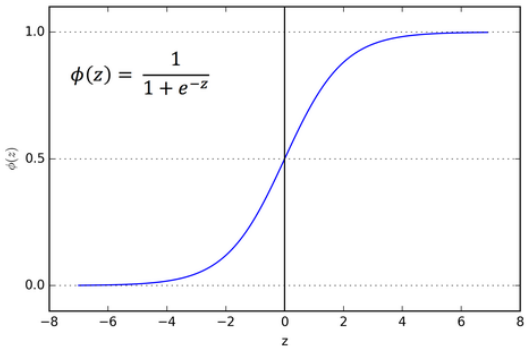



Figure 2.9: Graph of the Sigmoid Activation function. Image source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

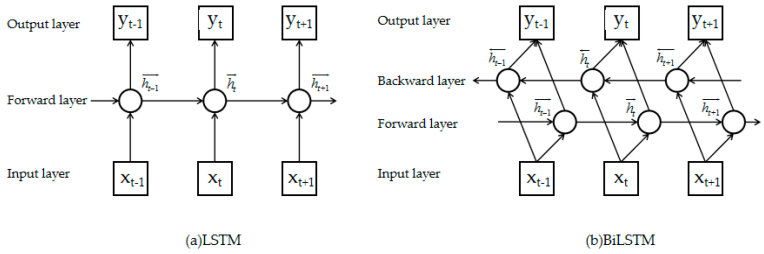



Figure 2.10: The architecture of the (a) LSTM model and (b) BiLSTM model [82].

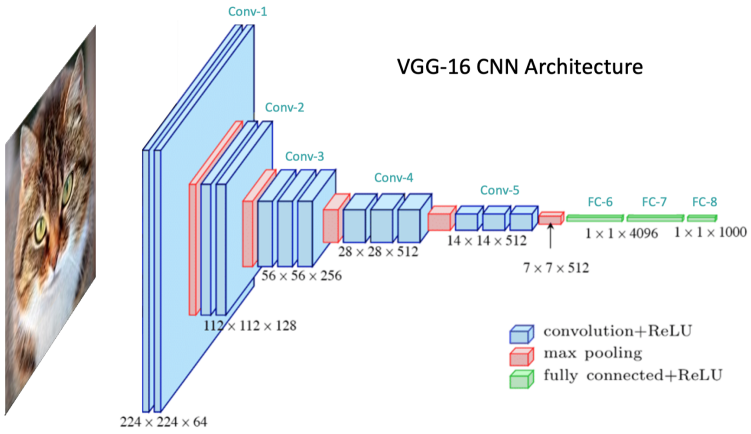



Figure 2.11: The VGG-16 Convolutional Neural Network (CNN) Architecture [62]. This depicts the main features of a CNN: convolutional layers, max pooling layers and a final set of fully-connected layers. Image source: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

or otherwise the output from the previous layer, called a hidden representation.

Convolution functions by the filter being slid across the input array. The scalar product between the filter and the current image region is calculated and placed at the centre point of the filter within the image.

The output, $C$, of a convolutional layer at the position $(a,b)$ is thus given as

$$C_{a,b} = \sum_{i=0}^{f-1} \sum_{j=0}^{f-1} w_{i,j} \cdot X_{i+a,j+b},$$

where $X$ is the input, of size $N \times N$, and $w$ is the filter, of size $f \times f$.

Presented here is 2D convolution, however, there also exists 1D and 3D convolutional layers, the difference being the filter and data size. 3D convolution is typically used for video data and requires 3D matrices of filters.

Max pooling layers are simpler, instead being used to reduce the size of a hidden representation within a model [48]. Max pooling layers similarly convolve across the output of the previous ANN layer, taking the maximum value in each pooling region to be the output, thus capturing the strongest activations [73]. This effectively reduces the effect of noise and reduces the size of data whilst remaining computationally efficient.

The output of a max pooling layer, $M$, at the position $(i,j)$ and the size of the output,

$m$, are defined as

$$M_{i,j} = max(X_{i-(f/2),j-(f/2)}, ...X_{i+(f/2),j+(f/2)}),$$

$$m = \frac{N-f}{s} + 1,$$

where $s$ is the stride size: how many positions the filter is moved in each operation [20].

A fully-connected layer is classified by its neurons. Every neuron of a fully-connected layer is connected to every single neuron from the previous layer [48].

Overall, CNNs are fantastic to better visualise and identify possible problems models may have with the underlying data thus forming useful, general models to be used for feature extraction or otherwise [81]. CNNs have proven themselves time and time again as excellent architectures for visual-based models.

### 2.3.6 Attention

Attention is a modern and popular method for speech recognition and language-based problems [42]. It offers fantastic performance for machine translation and is capable of understanding longer utterances than those present within the training data [11]. Some similar work, with attention for lip reading, has even used this to great success in the past [76].

Attention mimics how humans form information: by selectively building connections between different parts of information, rather than considering the entire context at once [47]. The major benefit of this, for lip reading, is that it allows for ignoring speech disfluencies, mapping varying numbers of states within the input sequence to the output sequence and focusing on crucial details within a sequence.

To calculate attention, there are three main components: the key ($K$), the query ($Q$), and the value ($V$). $K$ encodes the source data feature, $Q$ is a task-related representation (often the output we want to get) and $V$ is a new data feature representation [47]. $V$ is usually an alternate representation of the original sequence or even just the same as $K$ [47].

Here the standard, scaled dot-product attention will be used, rather than the other forms

of attention. This is the default offered by the Keras library[15]. This is defined as

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{|K|}}) \cdot V,$$

where $Q$ is the query, $K$ is the key and $V$ is the value.
Self attention is defined as attention where $Q$, $K$ and $V$ are all the same sequence. This method offers the advantage of self-alignment, enabling the identification of the importance of particular states within a sequence in relation to other states [66].

### 2.3.7 Stochastic Gradient Descent

Training an ANN is an iterative optimisation problem, where the goal is to find the inputs that minimise the output of an objective function [26]. At present, gradient descent is the most popular method for solving such a problem. Gradient descent, represented in Figure 2.12, uses the first-order derivative to make small, iterative steps in the opposite direction to the gradient of the loss function. This leads to the minima and the most optimal ANN weights for a given problem.
Stochastic mini-batch gradient descent is a variation of this process, where each update utilises a random subset of data samples to adjust the network weights [26]. This is opposed to just using a single data sample or the entire dataset (stochastic and batch gradient descent, respectively). Mini-batch gradient descent boasts several advantages over other methods, such as reduced computational cost, reduced noise and overfitting avoidance [26].
Two parameters crucial to SGD are Learning Rate and batch size.

#### 2.3.7.1 Learning Rate

Learning Rate (LR) relates to the size of steps within SGD. Finding the optimal LR is a difficult task requiring tuning [80]. As show in Figure 2.13, electing too high an LR could prevent any minima ever being found, whilst too low could slow training significantly and find local minima [80]. LR is one of the leading causes of overfitting within ML, so is a crucial design decision [36].

Recently multiple methods have been developed to vary LR over time, rather than

[15] https://keras.io/api/layers/attention_layers/attention/

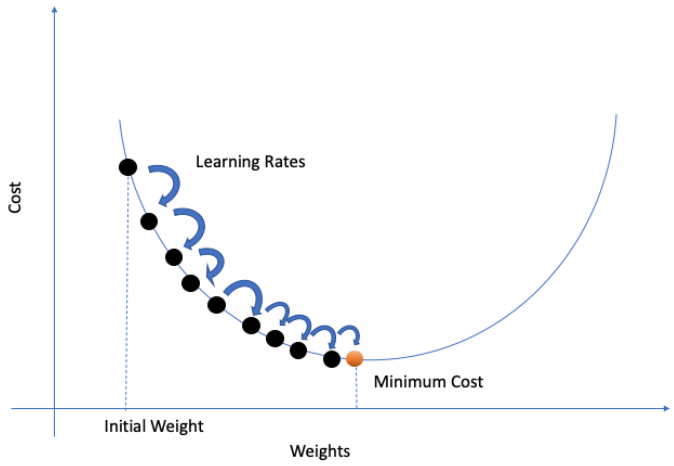



Figure 2.12: Stochastic Gradient Descent [21]. Here we can see how cost, or loss, is reduced via Stochastic Gradient Descent. This is the ideal for training, where we want to optimise our weights to achieve as small a loss as possible and so an optimal model.

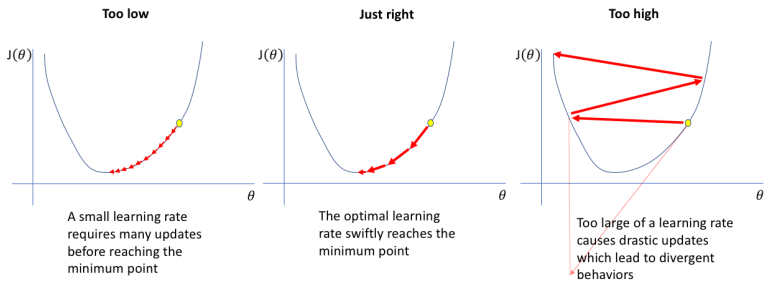



Figure 2.13: The effects varying LR. This shows how too high or low LR can cause problems when minimising a loss function. Image source: https://www.jeremyjordan.me/nn-learning-rate/.

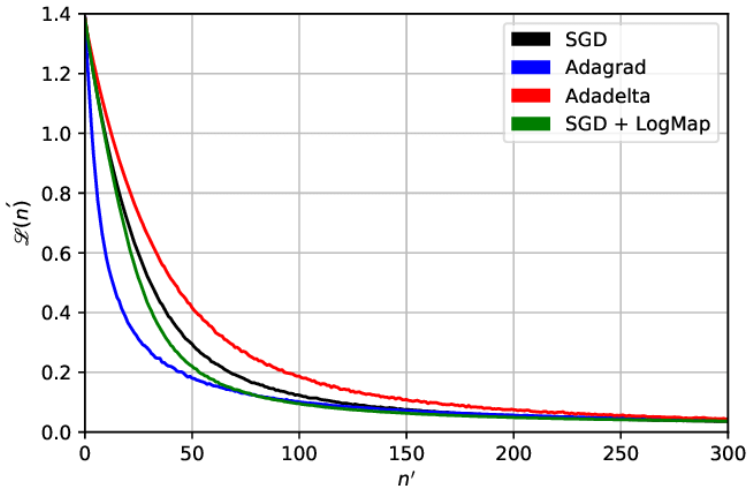



Figure 2.14: The advantage of Adagrad with the adaptive LR over other LR methods [31]. This shows the impact of utilising Adagrad, an adaptive LR scheduler over regular SGD. Here the same loss was achieved in far fewer epochs. The choice of LR scheduler is crucial as it could produce worse results, such as with Adadelta.

keeping it static throughout training [75]. Some of the different dynamic LRs are shown in Figure 2.14. Adaptive LR can save computational time whilst still finding the global minimum: the best of both stochastic and batch gradient descent.
The choice of LR scheduler, or optimiser, is incredibly important, possibly increasing or decreasing the number of epochs required for training. Some examples include linear decay, Adadelta, Adam and exponential decay [75].

#### 2.3.7.2 Batch Size

Batch size is the number of samples used within one training pass of a neural network [25]. Difficult to tune, batch size dramatically impacts the performance of neural networks.
Various research has been done into the impact of batch size, concluding that, for image classification, a higher batch size produces better performance [53]. However, this is not always the case. The option of batch size depends on the task being carried out and the other hyperparameters within the experiment [25]. With a large LR it is better to use a large batch size, and for a small LR, a small batch size is better [25].
This is due to the roles of batch size and LR. With a high LR we have that the model changes dramatically based on the data but if we have a small batch size and little data,

the model may become biased towards just these samples. Models could overfit, becoming too focused on the small dataset given. Alternatively, with a low LR and high batch size, the model may not learn any new information or change very much.
Another side effect of batch size is computational cost and efficiency. Larger batch sizes necessitate more calculations thus incurring increased computational costs [53]. However, larger batch sizes can increase computational efficiency attributed to better data and operation parallelism [14].
To optimise the training process, a high LR and batch size should be used initially, reducing these values throughout until only finetuning is required [25]. This balances computational cost and model efficiency. Suggested values for the batch size are around 32 samples and, therefore, this will be used within the experiments unless a more suitable size is identified.

### 2.3.8 Connectionist Temporal Classification Loss

Connectionist Temporal Classification (CTC) [22] is a loss metric utilised for sequencing labelling problems where the alignment between the sequence and labels is unknown [45]. The type of alignment found using CTC loss is depicted within Figure 2.15. CTC loss is used for various speech recognition problems including ASR and lip reading, offering excellent performance [74, 43].
CTC loss automatically learns connections between sequence frames and labels, removing the need for individual frame labelling [45]. Furthermore, compared with other methods, CTC loss produces models that are smaller and faster [33]. Much work has been done within this area and so this method will additionally be incorporated and compared with others for lip reading.
For lip reading, CTC loss will take the form of finding the most likely words (or letters, phonemes or visemes) given the input video feed.
First proposed in 2006, CTC loss aims to find the most likely label sequence, $Y = (y_1, y_2, ..., y_U)$ for a given input sequence, $X = (x_1, x_2, ..., x_T)$ [22]. Here, $y(u)$ is the target symbol at time step $u$ and $x(t)$ is the input state at time step $t$. So, we can define the output sequence, $Y^*$ as

$$Y^* = \underset{Y}{\operatorname{argmax}} P(Y|X)$$

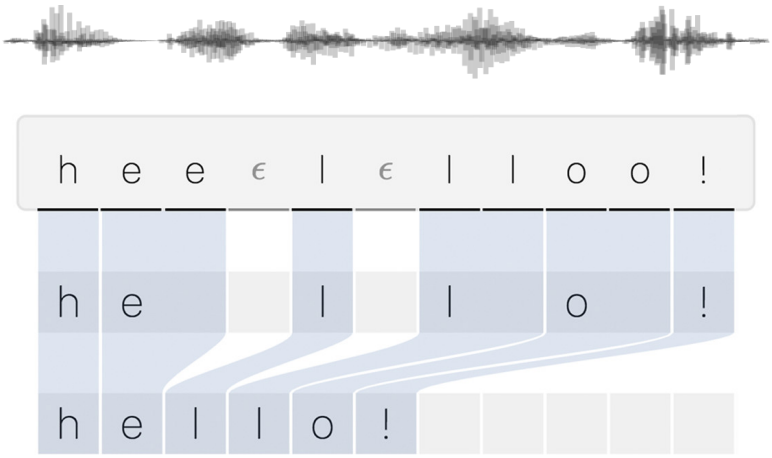



Figure 2.15: CTC Loss on an audio sequence. This shows how CTC loss can be used to predict a sequence of tokens which can then be merged into spoken information. Image source: https://distill.pub/2017/ctc/.

This probability is simplified to

$$P(Y|X) = \Sigma_{A \in X,Y} P(A|X)$$

$$P(Y|X) = \Sigma_{A \in X,Y} \prod_{t=1}^{T} P_t(a_t|X)$$

$$\therefore Y^* = \underset{Y}{\operatorname{argmax}} \Sigma_{A \in X,Y} \prod_{t=1}^{T} P_t(a_t|X)$$

Where $A$ represents the alignments from the input to output sequence and $a_t$ the alignment at time step $t$ [22].
This can then be used to train the model's parameters. By trying to maximise the likelihood of the correct $Y^*$ or minimising the negative log-likelihood (shown below), we can optimise the model's parameters [22].

$$CTC\ Loss = -\Sigma_{(X,Y)} log P(Y|X)$$

Training using CTC loss is computationally expensive, as finding the optimal alignments, requires dynamic programming [22].

## 2.4 Related Work

It is important to understand the accomplishments and failures of past work, in the field of automated lip reading, to inform a way forwards.

One of the most famous lip reading solutions was proposed by Chung et al. [12] in 2016. Chung et al. generated a huge dataset of word utterances from BBC[16] broadcasts, now referred to as LRW. This was key to the development of lip reading models and made it possible for others to build upon their work. Their published dataset became widely used due to its easy access and high quality.

They proposed a series of different CNN based architectures for lip reading, which exceeded the public performance benchmark at the time. Furthermore, they speculated about utilising LSTM units to better understand full sentences, rather than just single utterances. This forms a good place to start training.

In 2020, Luo et al. [41] utilised an encoder and bidirectional decoder architecture to carry out multilingual lip reading by classifying phonemes. This research presented state-of-the-art performance within two benchmarks: English and Mandarin lip reading. Luo et al. proved the usefulness of phonemes, especially across different languages, and the benefit of bi-directional learning.

Just before this work commenced, in 2023 Xue et al. [76] proposed LipFormer, a sequence-to-sequence Transformer that utilised self and cross attention, GRU units and a combination of visual and landmark features. LipFormer was proposed for the task of visual-only Mandarin Chinese lip reading and could generate both the Pinyin and Chinese characters for a given sequence. This work confirmed that lip landmarks help lip reading and can reduce the Word Error Rate (WER) of this task [76].

[16] https://www.bbc.co.uk/

# Chapter 3

# Design and Implementation

## 3.1 Requirements

The following requirements are necessary to achieve the objectives outlined within Section 1.4:

1. Gain access to a suitable dataset
2. Create a data generation pipeline to preprocess data into a usable form
3. Find suitable hardware to accelerate data generation and model training
4. Train various ANN networks in a valid and reliable experiment
5. Contrast the results in an objective way
6. Produce a GUI to showcase the models produced and investigate model improvement

## 3.2 Hardware Acceleration

To train a large ANN, within a concise time, state-of-the-art GPUs are required. There are two primary options for hardware acceleration to speed up training:

- Google Colaboratory[1]
- University of Manchester (UoM) provided systems such as the CSF[2]

[1] https://colab.research.google.com/
[2] https://research-it.manchester.ac.uk/services/the-computational-shared-facility-csf/

This section will analyse and compare these two options, justifying the choice of using the CSF.

### 3.2.1 Google Colaboratory

Google Colaboratory (or Google Colab) is an online Juypter notebook development platform offering multiple stages of hardware acceleration.
Colab offers great ease of use, not requiring a Google account and providing a degree of free use [7]. Other benefits of this method include its control over the level of hardware acceleration, easy package installation and various extensions, such as increasing levels of data storage, to complement large projects.

Despite its popularity, Google Colaboratory is not flawless. The biggest issue with this software is its steep pricing and quality at certain subscription levels. This section will analyse the question: are these levels worth the money?
As shown in Figure 3.1, there are four primary subscription levels. This analysis will not include "Colab Enterprise" as this is typically used by larger bodies such as whole companies or research groups.
The free version of Colab, not shown in Figure 3.1, allows execution of small Python programs with a low power GPU. One major problem with this level is that access to this acceleration is limited by an unspecified number of computation units. After you have used up your units, you are blocked from utilising this GPU again. Furthermore, a lack of background execution makes this subscription level unstable, with cache eviction causing the webpage to lose the current runtime, along with potential training runs and results. Altogether, this makes the free level of Colab limited, unstable and not suitable for training large ML models.
The "Pay As You Go" level gives limited access to more powerful GPUs. The amount of compute units given at this level is often used quickly, especially at the beginning of research when many different things are being trialled. Moreover, Google provides no recovery for interrupted runtimes, meaning that if credits run out halfway through a training run, results can be lost. This results in high money wastage.
The "Colab Pro" level, kindly marked as recommended by Google, is an evil twin to the "Pay As You Go" level. Although it provides similarly powerful GPUs, more memory for running sessions and a terminal, it locks users into a subscription. The extra memory is a useful extension as notebooks can occasionally fill up RAM if too much data is loaded or variables are not cleared.
"Colab Pro" contains all the disadvantages of "Pay As You Go" and the benefits are

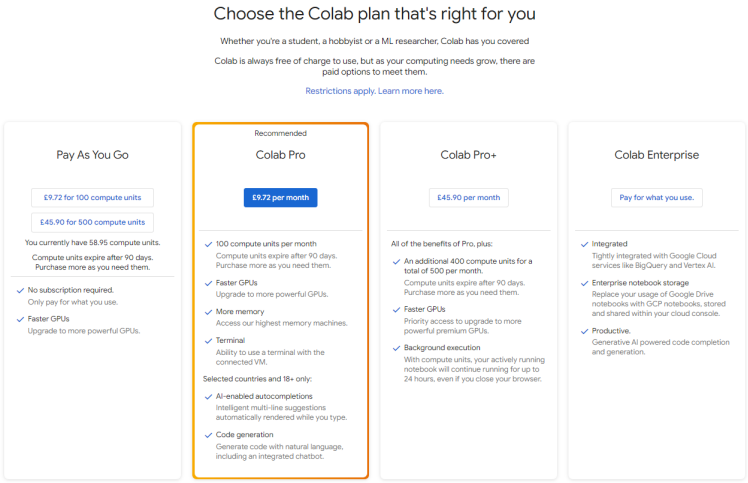



Figure 3.1: A screen-grab of the Google Colab subscription pricing as of March 2024. Image source: https://colab.research.google.com/signup.

not significant enough to constitute entering into a subscription. Google does at least make it easy to cancel subscriptions[3].

I recommend utilising "Colab Pro+" as opposed to "Colab Pro". This subscription level gives a huge 500 compute units, access to the fastest GPU acceleration and background execution. The pricing of this level also saves at least £2.7 compared to "Pay As You Go". Background execution saves money, time and effort by preventing runtimes from dying. However, this pricing is still very high, especially if this amount of credits is not required.

Overall, Google Colab is a double-edged sword; it is a useful tool, allowing fast development to those without access to hardware acceleration, however, it is an expensive option. Google Colab will not be used for this project due to the pricing and the alternative option of the CSF.

### 3.2.2 Computational Shared Facility

The University of Manchester (UoM) [4], provides a high-performance computing cluster dubbed the Computational Shared Facility (CSF) [5]. This is provided for primarily staff and PhD students to carry out computationally intensive tasks or jobs requiring possibly hundreds of cores simultaneously.

[3] https://colab.research.google.com/cancel-subscription
[4] https://www.manchester.ac.uk/
[5] https://research-it.manchester.ac.uk/services/the-computational-shared-facility-csf/

The CSF is a free-to-use resource that staff and students can request access to. It is incredibly fast, consisting of thousands of CPU cores and 100 GPUs such as Nvidia[6] V100 Volta and A100 Ampere GPUs.
Despite the various possible benefits, many students decide not to use the CSF due to its steep learning curve, batch job system and on-campus requirement.
The CSF is a batch job system, requiring users to submit text files of requested commands and wait for the result. Figure 3.2 shows an example of the job queue for the CSF. Below are listed some of the crucial commands for the CSF:

- **qsub *job_script.txt***: Submit a job file to be run
- **qstat**: See the list of jobs you have submitted. They can be listed as *qw* (waiting in the job queue to run), *r* (running) or *Eqw* (Job has errored and will wait forever). If the job has already been completed then this command will show an empty queue
- **qdel *id***: Stop and delete the job with the ID specified

Some major hurdles encountered with the CSF during this project were:

- Job files were prepared in Windows, causing an issue. Job files had to be prepared in a Linux format or converted using the *dos2unix* command
- Running Juypter Notebook[7] files automatically required the use of the command *jupyter nbconvert*
- Conda environments[8] where required to install the correct versions of specific packages
- The correct and compatible versions of Cuda and TensorFlow[9] required installation within this environment

[6] https://www.nvidia.com/en-gb/
[7] https://jupyter.org/
[8] https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html
[9] https://www.tensorflow.org/install/source

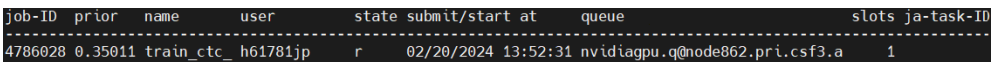



Figure 3.2: A screen-grab from MobaXTerm of a CSF job running. Here the *qstat* command has been used to see the running jobs submitted from the current user. One job is running currently, titled "train_ctc_based.txt".

The code in Listing 3.1 shows an example of a CSF job script. The script is used to automatically run Juypter notebooks for training.

```
#!/bin/bash --login
#$ -cwd

#$ -l nvidia_a100=1
module load apps/binapps/jupyter-notebook/6.0.0 #Load Juypter module

source activate myenv #Activate Conda env

#Install correct Cuda and Tensorflow
conda install -y -c conda-forge cudatoolkit=11.8.0
pip install --isolated nvidia-cudnn-cu11==8.6.0.163 tensorflow
    ==2.13.*
conda install -c nvidia cuda-nvcc --yes

#Configure based on information from UoM
CUDNN_PATH=$(dirname $(python -c "import nvidia.cudnn;print(nvidia.
    cudnn.__file__)"))
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib:
    $CUDNN_PATH/lib
export XLA_FLAGS=--xla_gpu_cuda_data_dir=$CONDA_PREFIX/lib

#Run the Juypter notebook experiment
jupyter nbconvert --execute --to notebook --ExecutePreprocessor.
    timeout=-1 --inplace proj_code/colabs/Train\ Video\ Based.ipynb

source deactivate
```

Listing 3.1: Example of a CSF job script. Here the CSF Juypter module is loaded: this allows Juypter-based commands to be run. Then, the Conda environment is activated and the correct packages are installed and configured, based on debug information from the University. The notebook is then run before the environment is deactivated again.

The CSF was utilised for this project. The CSF provides 4–5 GB of RAM for free (with more requiring only a request) compared with Google Colab's 15 GB with more requiring a further subscription. The CSF does have a steep learning curve but it is reliable, scalable, efficient and free compared to Google Colab. The CSF provides the benefits of "Colab Pro+" without the extreme pricing.
An SSH connection was required to access the CSF and submit jobs. However, due

to a recent cyber incident, remote access to this network was suspended. Therefore all work was conducted on-campus via the MobaXterm[10] application. This allowed for easy control over SHH connections.

## 3.3 Data Generation Pipeline

This section will outline the process used to generate useful data for training lip reading models and the source dataset selected for training. It will delve into the design decisions and processes employed for data preprocessing.

The general process for data generation is shown in Figure 3.3. MediaPipe was applied to each video from the dataset extracting facial landmarks which enabled a crop of the mouth to be taken. Feature extraction was done on the lip crops and the lip landmarks were normalised. This data was stored along with the word, phoneme and viseme utterances for the video. Data was split into three sets: training, testing and validation.

The overall data generation pipeline can be broken down into four primary stages:

1. Landmark feature extraction, outlined in Section 3.3.2.1
2. Visual feature extraction, outlined in Section 3.3.2.2
3. Word utterance processing, outlined in Section 3.3.3
4. Random data splitting, outlined in Section 3.3.4

### 3.3.1 Dataset

For all of these experiments the same dataset and data split were maintained, ensuring consistent results and reliable comparison between the different models.

Shown in Figure 3.4, the choice of datasets was primarily between that of LRW, LRS2, LRS3-TED, LRW-1000 and GLips. The final two of these were not considered because they are transcribed in alternative languages and therefore would pose an additional layer of complexity. Furthermore, LRS3-TED was not considered as the data is not available; the link[11] to this data leads to a 404 error.

The choice between LRW and LRS2 depends primarily on the task being carried out.

[10] https://mobaxterm.mobatek.net/

[11] https://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrs3.html

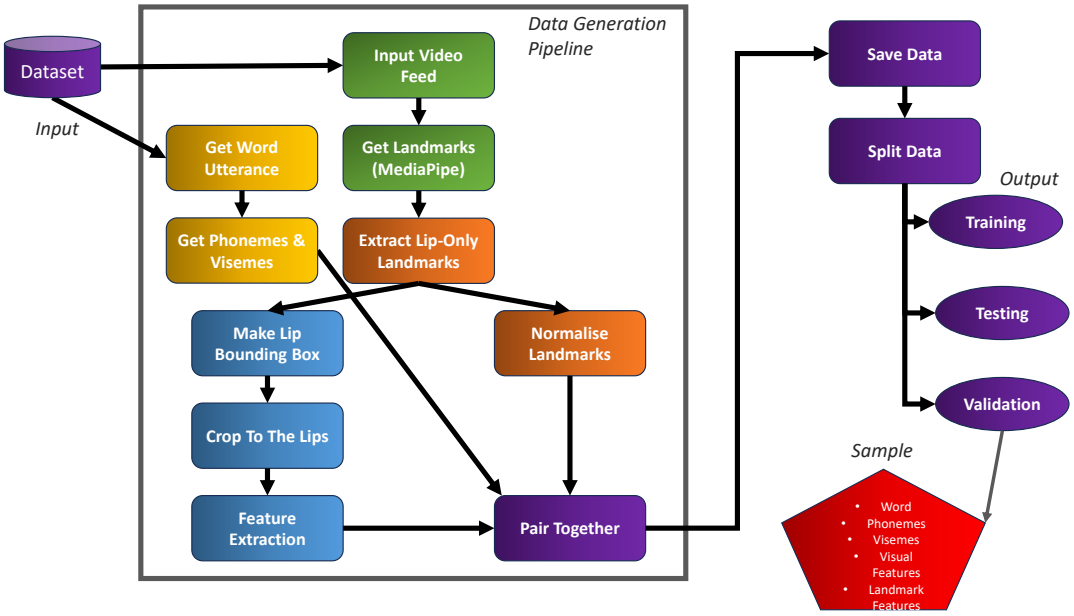



Figure 3.3: The Data Generation Pipeline from the LRW data to split samples, ready for training. This shows how data samples are constructed for training, combining the labels, visual features and landmark features. All data is preprocessed and saved before a data split is subsequently done.

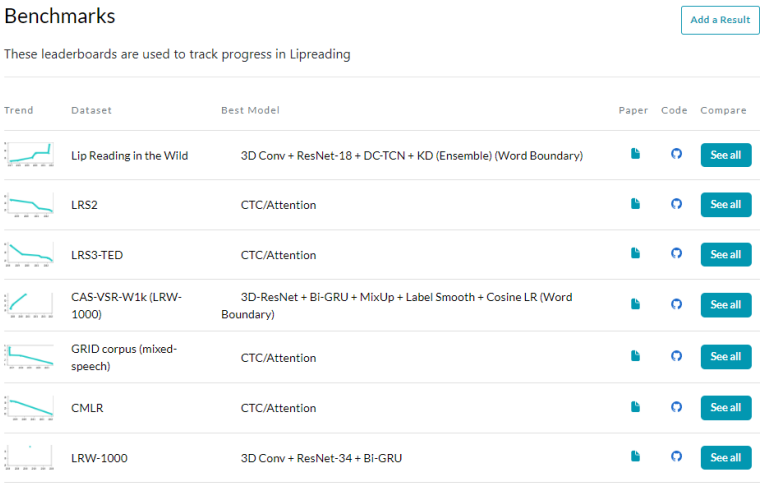



Figure 3.4: The benchmark for lip reading via ML. This shows some of the available datasets available for training lip reading models. Image source: https://paperswithcode.com/task/lipreading

Both of these datasets are available upon request online[12]. LRW was selected because its structure suits the task being carried out.

As shown in Figure 3.5, LRW consisted of a set of folders named after the primary word utterance within each video. As shown in Figure 3.6, each video consisted of 29 frames from BBC broadcasts. During these clips, a primary word was uttered, but possibly multiple other words were also spoken either before or after the primary utterance.

This structure benefits the model as it provides a large amount of data for each different word utterance, allowing training to fully understand the structure and intricacies of each word. Furthermore, the addition of extra, unlabelled words within clips adds variation to the data and produces models capable of recognising single words within sentences. This is similar to the original research carried out by Chung et al. [12]. The primary downside to LRW is the possibility of overfitting due to the extra noisy, word utterances.

The experiments involved training models capable of understanding a small subset of words. LRW suited this better compared with LRS2 which has a collection of random videos, each with a sentence of labelled words. The diversity of words within LRS2 is useful but not for the task being carried out.

Four words were randomly selected for the models to be trained upon: *about*, *believe*, *chance* and *family*. These labels were transformed into one hot encodings for training. A subset of words was used to reduce training times and for model comparison. After preprocessing, 4,392 samples of these words were present within the dataset. This is slightly smaller than expected (4400) as MediaPipe could not find a valid face within some images, typically due to the orientation or clipping of faces within frames.

To gain access to the dataset, a request had to be sent to the BBC, explaining the work to be carried out. Once this request was approved, a login was provided for the website[13], allowing data to be downloaded.

LRW was split into seven parts, each approximately 10 GB in size. Once downloaded, these parts then had to be concatenated together to form a zip file, containing the full dataset. To combine the file sections the command below was run

```
% Linux Commands:
cat lrw-v1* > lrw-v1.tar
tar -xvf lrw-v1.tar

```

[12] https://www.robots.ox.ac.uk/~vgg/data/lip_reading/

[13] https://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrw1.html

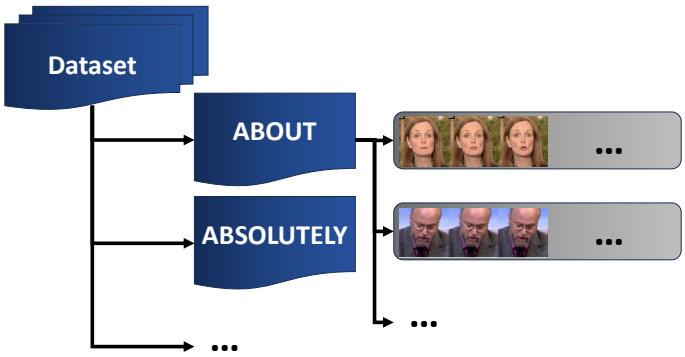



Figure 3.5: The structure of the LRW dataset [12]. LRW contains a set of directories labelled with the primary word being spoken. Each directory stores 1100 videos, each 29 frames long of BBC broadcasters saying a small set of words. The primary word spoken is within the middle of the video.

Figure 3.6: An extract from the LRW dataset [12]. This clip is labelled as "ABOUT" but it contains the words "worried about liability" with the final word half cut off.

```
% Windows Commands:
type lrw-v1* > lrw-v1.tar
```

Listing 3.2: Commands to combine the sections of the LRW dataset. Note here that there are two separate sets of commands. The first are the commands presented on the LRW website however, this is seemingly for Linux. The second set is for Windows.

At first these commands caused some issues. As the commands presented for the dataset are in Linux, and the development system utilised was Windows, some disparity was encountered. Running the commands above in Windows resulted in the file segments being repeatedly combined to form the zip file. After the set was concatenated once, the process was repeated seemingly infinitely.
Instead, a different command, shown in Figure 3.2, had to be run before 7-Zip[14] was employed to extract these files.

### 3.3.2 Data Preprocessing

To prepare the data, the videos within LRW directories specified as *about*, *believe*, *chance* and *family* were utilised. Each video within these directories was extracted and broken down into 29 individual frames. For each frame, MediaPipe was used to find landmark and visual features. The processes for these methods are detailed in the sections below. Some experiments involved just one of these data types whilst others used both.

#### 3.3.2.1 Landmark Feature Extraction

Explored in Section 2.2.2, MediaPipe provides a Face Landmark Detection[15] model. This model was used to generate a set of 478 facial landmarks for each frame of the data subset used. Forty of these landmarks, specified in Listing 3.3, corresponded to parts of the lips, so were extracted and used for training. Examples of this extraction process can be seen within Figure 3.7 and Figure 3.8
Landmarks are sets of $(X,Y,Z)$ coordinates relative to the position of facial landmarks within images. This is unsuitable for training as variation in the global position of faces within the video feeds can cause underfitting and make lip reading results incomparable. Therefore, normalisation was used to transform these coordinates into a

[14] https://www.7-zip.org/
[15] https://mediapipe-studio.webapps.google.com/demo/face_landmarker

Figure 3.7: An example of using MediaPipe to find the lip landmarks and lip region, from LRW data samples [12]. Here MediaPipe's Face Landmark Detection was used to generate facial landmarks which were then used to find and crop the lip region. The left images show the default image, the middle with lip-only landmarks highlighted and the right with the subsequent lip crop.

| Raw Landmark | |
|---|---|
| Axis | Coordinate |
| X | 0.61384851 |
| Y | 0.59772831 |
| Z | 0.01386063 |

| Normalised Landmark | |
|---|---|
| Axis | Coordinate |
| X | 1. |
| Y | 0.29053584 |
| Z | 0.01386063 |

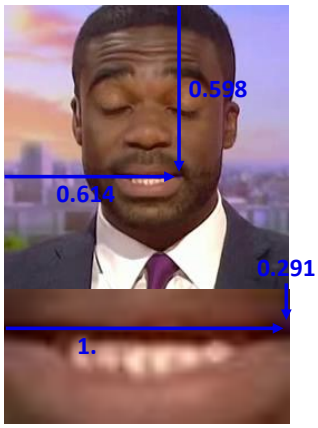



Figure 3.8: An example of a normalised MediaPipe landmark, from the LRW data samples [12]. Here MediaPipe is used to find facial landmarks. A single landmark is observed, which is displayed on the full image and the lip crop. The landmark is normalised based on the whole lip crop.

proportionate form.

```
lip_landmarks_outer_keys: typing.List[int] = [
    61, 185, 40, 39, 37, 0, 267, 269, 270, 409, 291, 375, 321, 405,
    314, 17, 84, 181, 91, 146,
]
lip_landmarks_inner_keys: typing.List[int] = [
    78, 191, 80, 81, 82, 13, 312, 311, 310, 415, 308, 324, 318, 402,
    317, 14, 87, 178, 88 95,
]
```

Listing 3.3: A list of the MediaPipe facial landmarks that correspond to the lips. These correspond to the indices visualised within Figure 2.4 as the facial landmarks associated with the lips.

The mathematical function for this normalisation is shown below

$$X_i = \frac{X_i - X_l}{w},$$

$$Y_i = \frac{Y_i - Y_l}{h},$$

$$Z = Z,$$

where $(X_l, Y_l, 0)$ corresponds to the upper left coordinate of a bounding box created using all of the lip landmarks (defined in Section 3.3.2.2), $(X_i, Y_i, 0)$ correspond to the coordinates of the $i$th lip landmark, and $w$ and $h$ are the width and height respectively of the lip bounding box in the image.
The width, $w$, and height, $h$, of the lip bounding boxes were formulated using the equations below

$$w = \max\{X_1, X_2, ...X_{40}\} - \min\{X_1, X_2, ...X_{40}\},$$

$$h = \max\{Y_1, Y_2, ...Y_{40}\} - \min\{Y_1, Y_2, ...Y_{40}\}.$$

The $Z$ coordinate was not normalised as this represents the depth of a landmark within an image. Thus, it is less useful for training and not relative to the image coordinates. Once the landmarks were normalised they were saved to the preprocessed dataset.

#### 3.3.2.2 Visual Feature Extraction

Landmark features are not enough on their own. There is substantial, relevant data related to speech that is not captured by the placement of the lips alone. For example, the teeth and tongue are both key to distinguishing certain phonemes yet not captured by MediaPipe. Utilising wider visual features is a technique adopted by various other successful lip reading systems [4] and has been proven to produce better results [76].
To produce visual features, the lip landmarks extracted in the previous stage were reused to create a bounding box around the lips. This bounding box was used to take a crop of each frame of the videos from LRW.
To find the bounding box, defined by the coordinates $(X_l, Y_l)$ (for the top left corner) and $(X_r, Y_r)$ (for the bottom right corner), the following calculations were carried out:

$$X_l = \min(X_1, X_2, ...X_{40})$$

$$Y_l = \min(Y_1, Y_2, ...Y_{40})$$

$$X_r = \max(X_1, X_2, ...X_{40})$$

$$Y_r = \max(Y_1, Y_2, ...Y_{40})$$

Where $X_i$ defines the $X$ coordinate of lip landmark $i$ and $Y_i$ defines the $Y$ coordinate of lip landmark $i$.
InceptionV3 Imagenet [16] [65] was used for visual feature extraction of the lip crops. InceptionV3 Imagenet is lightweight and has high performance, making it useful for devices that might not use a GPU, whilst still boasting very high accuracy [65]. It has a straightforward implementation via Keras. Possible future work could aim at experimenting with alternative feature extractors to compare performance metrics.
As shown in Figure 3.9, lip crops were passed through the InceptionV3 Imagenet feature extractor, generating 2048 long feature vectors. Visual features were paired with the corresponding landmark features before being saved.

### 3.3.3 Word Utterance Processing

During training, videos were classified into a set of different classes:

- The **words** spoken: multi-class classification and single-label classification

[16] https://keras.io/api/applications/inceptionv3/

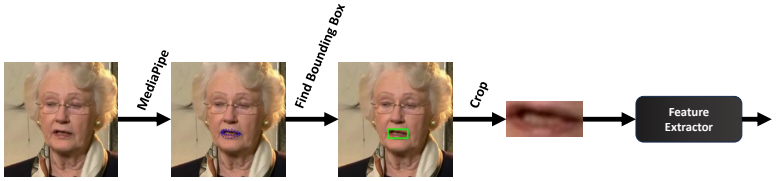



Figure 3.9: The process of visual feature extraction. MediaPipe is used to find lip landmarks. A bounding box is made surrounding all landmarks before a crop is taken. The crop is passed into a feature extractor.

- The **letters** of words spoken: multi-class classification and multi-label classification
- The **phonemes** uttered: multi-class classification and multi-label classification
- The **viseme**s expressed: multi-class classification and multi-label classification

LRW is currently annotated with only the words (and, therefore, also the letters) spoken within videos. Thus, both the phonemes and visemes for these had to be generated automatically.

To produce the phonemes, from words, the Python library NLTK[17] was used. Described in Section 2.2.4, this library can be used to convert words into their constituent phonemes via the Carnegie Mellon Pronouncing Dictionary (cmudict).

As explained more in Section 2.1, retrieving the visemes of words is a harder task. Classification based on visemes is potentially a more fruitful endeavour for classification; visemes relating more closely to the visual aspects of language compared with letters or phonemes. However, there currently exists no defined viseme set, let alone a tool for Phoneme to Viseme (P2V) conversion.

For this set of experiments, as suggested by Bear et al. [6], the P2V mapping proposed by Lee et al. [34] was employed for P2V conversion. This is shown within Figure 3.10. Note that some of the phonemes have two overlapping visemes (such as "EY" and "AW"). Although Lee et al. [34] do not elaborate upon this, the reason for this is likely that some phonemes required multiple visemes to express.

Phonemes were converted into visemes using a Python dictionary. Once words had equivalent phonetic and visual forms, these data labels were exported with the corresponding data samples.

[17] https://www.nltk.org/_modules/nltk/corpus/reader/cmudict.html

| Viseme | Phonemes |
|---|---|
| p | B, P, M, |
| t | D, T, S, Z, TH, DH, |
| k | G, K, N, NG, L, Y |
| ch | JH, CH, SH, ZH, |
| f | F, V |
| w | R, W |
| iy | IY, IH, |
| eh | EH, AE, |
| eh, iy | EY, |
| aa | AA |
| aa,uh | AW |
| aa,iy | AY |
| ah | AH |
| ao | AO |
| ao, iy | OY |
| ao, uh | OW |
| uh | UH, UW |
| er | ER |

Figure 3.10: The adopted P2V mapping, as proposed by Lee et al. [34] and promoted by Bear et al. [6]. This shows which phonemes fall into each viseme group. Some phonemes are made up of multiple visemes and are therefore marked as such. There are thirteen visemes in total (and an additional blank viseme, although this has not been added above).

### 3.3.4 Data Splitting

Data splitting refers to the process of separating data into multiple sets, one to train the ANN and another to evaluate it [51].
There are three typical sets for data splitting: training, testing and validation. The training set is used to train and optimise the model's parameters. The validation and testing sets are used to provide an unbiased view of the model's performance, kept separate to prevent overfitting. The validation set is used to evaluate the model's performance and fine-tune its hyperparameters throughout training. The testing set is used to evaluate the model's final, overall performance and to directly compare different models.
There are various methods of data splitting, including but not limited to Simple Random Sampling (SRS), trial-and-error and systematic sampling [56]. These methods for data splitting primarily affect the variance of the model error. For example, trial-and-error sampling estimates the model error with lower variance than other methods [56]. For this research, the simple and easy-to-implement SRS was utilised during training, testing and validation even despite its high variance.
As shown in Figure 3.11, there are different proportions of data split. Here data was split by randomly shuffling the data samples and selecting the first 80% of samples for training, the next 10% for testing and the final 10% for validation. This is a common data split used within ML projects.

## 3.4 Design of Experiments

This section will cover the structure of experiments covered within this research, including the many different design decisions and principles followed.

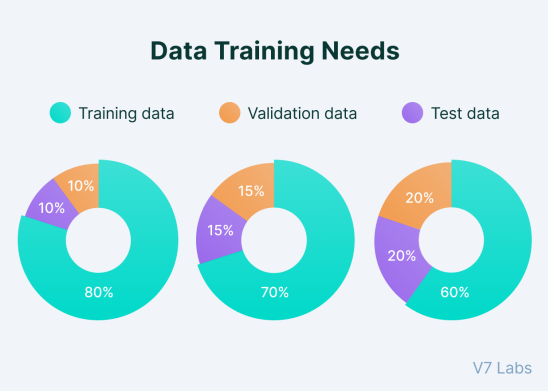



Figure 3.11: A diagram of common data split ratios used in Machine Learning (ML).

### 3.4.1 Experiment Structure

The experiment structure is a key design decision that will inform the reliability of results and help to better compare alternative lip reading architectures. This structure is depicted in Figure 3.12.
After data was collated and preprocessed, a set of experiments were carried out. The ANN was repeatedly altered, trained and evaluated to study the impact that various hyperparameters had on lip reading.
Models were trained using the training and validation sets, employing early stopping as specified within Section 3.4.2. For each experiment, the optimal models were loaded back in and run against the testing dataset. This was used to objectively capture the performance on unseen data. The accuracy, loss and Word Error Rate (WER) (as defined in Section 2.2.5) were observed.
Models were compared via their validation and testing metrics to find the best hyperparameter settings and draw conclusions as to the difficulties or benefits of different methods.
There were three principles fostered for these experiments:

- Maintain the data split
- Make single changes between experiments — changes can be large or small
- Maintain optimal changes to produce increasingly optimal models

The first concern is that of data. To allow for better model comparison, the same data sets should be employed to train and then evaluate the different models. This makes them directly comparable. Therefore, the data split algorithm was incorporated into the data generation pipeline, maintaining the split between the different experiments. This had the benefit of saving computation and processing by avoiding repeated, redundant data splits. The data split was made on each word class individually rather than the entire dataset. This ensured similar quantities of each data class for training, testing and validation.
The next concern is how to compare different experiments. Typically, only one thing should be varied between experiments to ensure that any changes in performance can be attributed directly to the change made. This is a hard task for ML, where many different parameters are at play. Furthermore, this is made difficult by the definition of what a “single change” entails: is this a single parameter change or a whole different ANN architecture? Early stopping, explained further in Section 3.4.2, means that

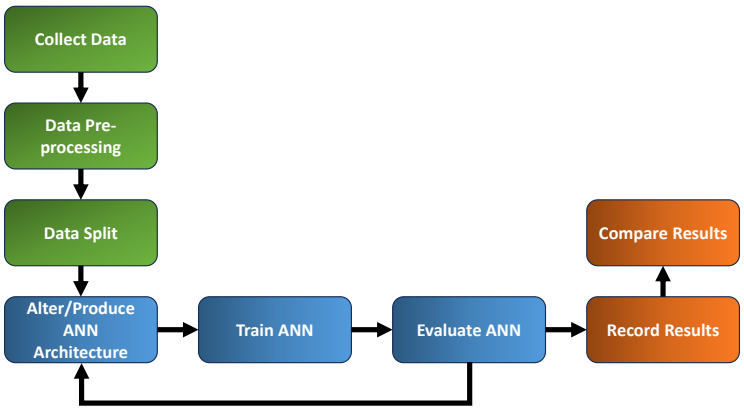



Figure 3.12: The workflow adopted within the experiments. The green section shows the steps that were carried out first and only once. The blue section was repeated several times, for each experiment. This section shows the iterative nature of the experiments, where changes are repeatedly made to gather further results. The orange section shows the critical reflection within this research.

models could be trained for varying numbers of epochs which adds further variation. For this research, only one thing was purposefully altered, for example batch size or Learning Rate (LR). Changes between experiments could be more extensive, such as trying a whole different model architecture, but these changes were fully documented for this reason. Variation due to early stopping could not be avoided compared with the potential benefit this method poses to the performance of models.

The final principle is maintaining optimal changes to models where possible. When beneficial changes were discovered, these were maintained between experiments. This ensured an increasing performance throughout.

### 3.4.2 Early Stopping

For all training runs, early stopping was employed to save the best model weights. This was achieved using Keras's callback, ModelCheckpoint[18]. This was configured to monitor the validation loss and save the model with the best performance. This would ensure the model is saved before overfitting occurs and metrics worsen.

[18] https://keras.io/api/callbacks/model_checkpoint/

### 3.4.3 Learning Rate

The Adam optimiser was used in conjunction with exponential decay during training. Exponential decay was utilised to decay the LR over time, as the local minimum was approached, thus scaling Adam's effect over time.
Adam, or adaptive moment estimation optimiser, is a modern LR scheduler made popular due to its effectiveness, simplicity and handling of large amounts of data [29]. In addition, Keras provides an easy-to-use Adam optimiser[19]. Adam works by selectively adjusting the LR for each parameter, based on the history of that parameter [29].
Exponential decay is a method of lowering LR over time, based on the current epoch. The benefit of using a decay is that a higher initial LR can be employed at first, preventing ANNs from learning noisy data. The LR is then reduced throughout, as the model converges to a global minimum in the loss. This reduces the negative impact on model performance during the early stages of training [79]. Exponential decay is a popular and simple method of this, as defined below

$$lr_{new} = lr_{initial} \cdot e^{-D \cdot n},$$

where $lr_{new}$ is the updated LR for the current epoch $n$, $lr_{initial}$ is the initial LR and $D$ is the decay rate. The decay rate is a further parameter introduced with exponential LR decay, used to control the rate of change.
In Keras, exponential decay[20] is configured with four different parameters. The "$decay\_rate$" and "$initial\_learning\_rate$" are easier to understand but for "$decay\_steps$" and "$staircase$" this is less the case. Keras gives the following function to define their formulation

$$LR = initial\_learning\_rate \times decay\_rate^{(step/decay\_steps)},$$

where $step$ is the number of total steps. This formulation is controlled by "$staircase$" which controls the type of division used within the exponential $(step/decay\_steps)$. When set to false, this formulation will be true division, resulting in a continuous change to the LR. But "$staircase$" is set to true, integer division is employed, resulting in a discrete, staircase-like change in the LR.
For the experiments that employ exponential decay, "$staircase$" will be set to false, resulting in this continuous change to the LR. The "$decay\_steps$" will be configured

[19] https://keras.io/api/optimizers/adam/
[20] https://keras.io/api/optimizers/learning_rate_schedules/exponential_decay/

based on the amount of data samples and the batch size. A formulation for this is defined below.

$$decay\_steps = \frac{total\_training\_samples}{batch\_size}.$$

This ensures that all samples are utilised exactly once during each training epoch.

## 3.5 Graphical User Interface

The Graphical User Interface (GUI) integrates the features of the data generation pipeline, outlined in Section 3.3, and a subset of the models trained in Chapter 4. It brings these elements together to better showcase the system as a whole and test the lip reading system in practise.

### 3.5.1 Inference Workflow

Shown in Figure 3.13, the GUI was constructed with a specific workflow for capturing images, processing them and performing inference. The workflow was used to capture the required information to pass into the lip reading models outlined in Chapter 4 and identify the words being spoken.
There are three main parts of this workflow

1. Data capture
2. Running the Artificial Neural Network
3. Displaying the results

For data capture a similar method was carried out as specified in Section 3.3. MediaPipe was employed to identify faces, generate landmarks and produce visual and landmark features.
Running the ANN was a more configurable process, controlled by the GUI's widgets. The visual and landmark data from data capture were passed into the ANN selected within the GUI.
The output of this was processed based on the selected model. The classes used were configured, as shown in Listing 3.4, based on how the different models were trained. For models that were trained to classify data into whole words, the word-based dictionary was selected to translate model output.

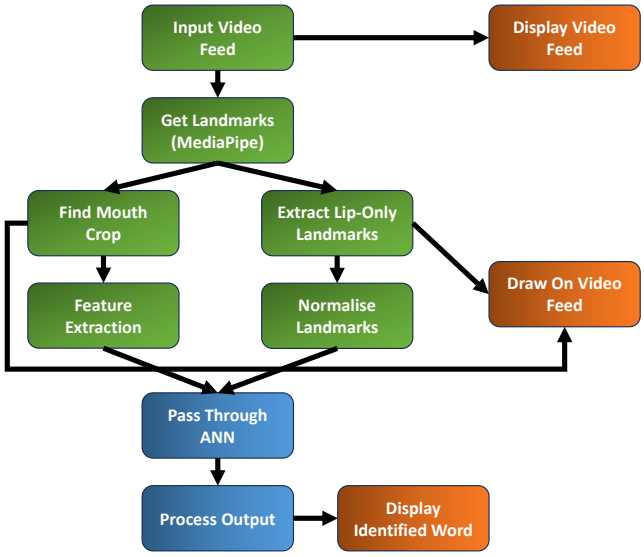



Figure 3.13: Inference workflow of the proposed lip reading GUI. The GUI will have a constant video stream. It will detect people, extract data and pass it through the optimised model. The identified words will then be displayed on the screen. Green boxes represent stages required for data capture, blue for running the ANN and orange for displaying the results.

```
word_based = {0: "ABOUT", 1: "BELIEVE", 2: "CHANCE", 3: "FAMILY"}
letter_based = {16: " ", 1: "A", 2: "B", 3: "C", 4: "E", 5: "F", 6:
    "H", 7: "I", 8: "L", 9: "M", 10: "N", 11: "O", 12: "T", 13: "U",
    14: "V", 15: "Y",}
phoneme_based = {1: 'AE1', 2: 'AH0', 3: 'AW1', 4: 'B', 5: 'CH', 6: '
    F', 7: 'IH0', 8: 'IY0', 9: 'IY1', 10: 'L', 11: 'M', 12: 'N', 13:
    'S', 14: 'T', 15: 'V', 16: ' '}
viseme_based = {1: 'aa', 2: 'ah', 3: 'ch', 4: 'eh', 5: 'f', 6: 'iy',
     7: 'k', 8: 'p', 9: 't', 10: 'uh', 11: ' '}
```

Listing 3.4: The label mappings used for the different models. This shows the mapping from the model output to the word labels for each different model output type.

However, a major design decision came in the form of configuring the input to models. The correspondence of frames to labels for some models was many-to-one, whilst for others it was one-to-one. Therefore, in addition to the label translation, each model included the number of frames required for a single prediction. For the many-to-one models, a buffer of data (the “frame buffer”) was maintained for each frame ready to be passed through the model at the correct time. Only once the frame buffer was full was a prediction made before the buffer was emptied, ready for new frames to be accepted.

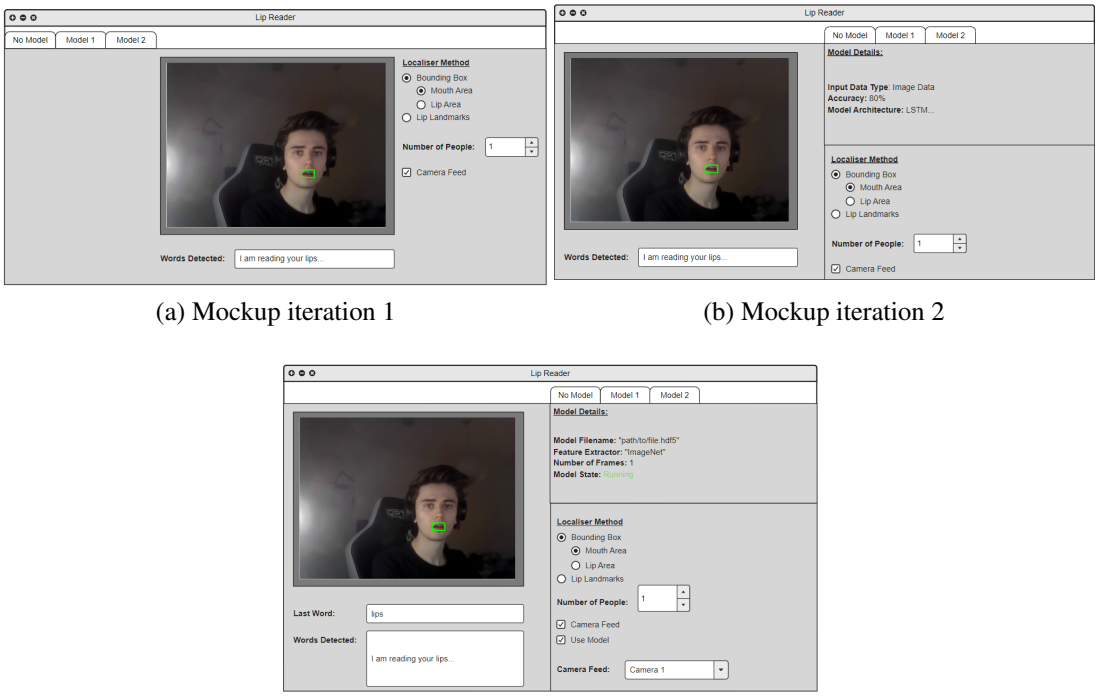
Displaying the results involved displaying the current view of the selected camera, displaying the landmarks or bounding box (according to user configuration) found using MediaPipe and displaying the processed output from the ANN.

### 3.5.2 System Structure

The design of the GUI went through three short iterations of mockups, as shown in Figure 3.14. All mockups were made using moqups[21]. After these mockups were developed, the main GUI was produced using Tkinter. The remaining time within this project permitted the GUI to be extended, adding an extra tab within the window for model fine-tuning. This is shown further in Figure 3.15. The features of the GUI are presented below:

1. **Mode Selector**: Used to change the interface between inference and training (or fine-tuning)
2. **Model Selector**: Used to change between different trained models. Only a subset of the models are provided to select between. Note the change here that a vertical layout was employed rather than horizontal. This was to allow for better distinction between the different models
3. **Model Properties**: This pane is used to give details about each of the different models
4. **Localiser Method**: This configures display feed annotation, drawing bounding boxes around the lip or mouth area, or displaying the lip landmarks. This process is outlined more in Sections 3.3.2.2 and 3.3.2.1
5. **Prediction Frame**: This displays the current prediction, the past predictions and the frame buffer length (as it fills up). This displays the current mouth state (open or closed), the process for which is explained in Item 10 and Item 11
6. **Camera Feed Selection**: Python was used to detect all available cameras connected to the current device. These were then presented in a drop-down box for selection
7. **Number of People in Frame**: Used to configure MediaPipe to detect the correct number of people in the frame. Utilised to display bounding boxes for the lips of several people

[21] https://moqups.com/

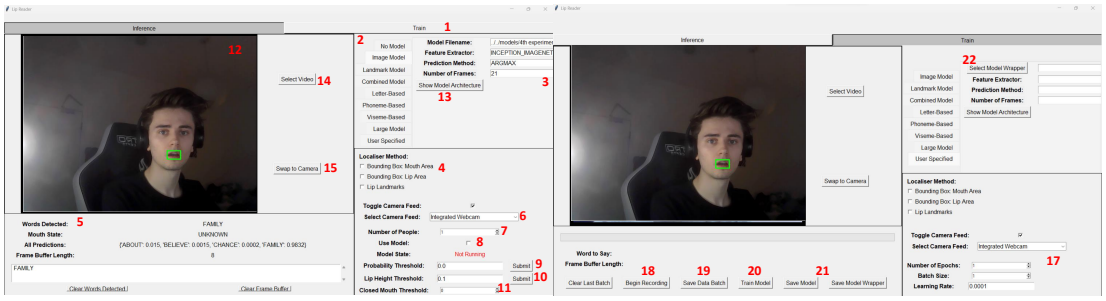



(a) Mockup iteration 1

(b) Mockup iteration 2

(c) Mockup iteration 3

Figure 3.14: The three mockup iterations of the Graphical User Interface (GUI). The first stage included changing between models, realtime video feed and control over localisation. The second altered mainly the look of the system. The final mockup added further controls for the model, mainly in the inference methods, and more important model details.




(a) Inference Tab

(b) Training (fine-tuning) Tab

Figure 3.15: The labelled Graphical User Interface (GUI). This labels each of the primary features of the fully implemented GUI. Here both tabs for the GUI are shown: inference and training (or rather, fine-tuning).

8. **Enable Model**: Used to begin running the model or pause it from making inference

9. **Probability Threshold**: The threshold for prediction probabilities. Predictions below this predictions will be set to *UNKNOWN*

10. **Lip Height Threshold**: The threshold for the distance between the average of the upper landmarks and the average of the lower landmarks. Below this threshold the lips are considered to be together, and therefore closed, for the current frame

11. **Closed Mouth Threshold**: A threshold for the number of frames in a row that the mouth is closed for. If the number exceeds this value then predictions will stop being made as the lips are considered as being closed. This saves computation by having the model switch off for a person who isn't speaking. When set to 0 this feature is not used to stop predictions

12. **Display Area**: This displays the current feed of the selected camera

13. **Model Architecture**: Clicking this button will open another window, displaying the model architecture of the currently selected model. This diagram is automatically generated in real-time using Keras util's plot_model[22] function

14. **Select Video**: This creates a file dialogue box, allowing selection of a video file. This file will then be repeatedly played in the window and used for inference

15. **Swap to Camera**: This swaps the feed back to the currently selected input stream

16. **Fine-tuning Configuration**: This area is used to modify and control the settings for model fine-tuning such as the LR, epochs and batch size

17. **Begin Recording**: Activates data collection, starting to collect frames from the input stream for the current word

18. **Save Data Batch**: This will temporarily save the data sample to be used for fine-tuning later

19. **Train Model**: This model starts fine-tuning of the model using the data

[22] https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model

20. **Model Saving**: These buttons are used to save the currently selected model or whole model wrapper to be loaded in and used through Item 21

21. **Load Model**: Selecting the "User Specified" model shows this feature. Clicking this button creates a dialogue, allowing a model wrapper file to be found and selected for use

One of the main, complex features of the GUI is that of stopping inference when the mouth is closed. This is controlled using Items 11 and 10. Item 11 controls the relative height between the upper and low lips. When this height is less than the threshold, the lips are considered closed. Item 10 controls inference associated with when the user has stopped speaking, stopping the model from predicting if it considers the speaker to have paused for a time.

Another feature worth further explanation is that of fine-tuning lip reading models. To fine-tune the model, a user is presented with the classes of the currently selected model. They can select Item 17 to begin recording data for fine-tuning and Item 18 to save that data internally. When Item 19 is selected fine-tuning of the current model will begin. This will utilise the training settings configured within Item 16 to fine-tune the current model. This could take some time depending on the settings. The resulting model metrics will be printed to the interface and the user can then opt to save the new model using Item 18.

# Chapter 4

# Experimentation and Results

This chapter presents various model architectures and their performance on lip reading. It contrasts them and explains the workflow to develop the best possible lip reading model. A summary of the experiments that will be performed is presented in Table 4.1.

| Experiment | Description |
|---|---|
| 1 | A basic CNN architecture feeding into an LSTM was trialled. Visual features were used as input |
| 2 | Data augmentation and frame pruning were utilised during pre-processing |
| 3 | A pure Bi-LSTM architecture was swapped in and assessed |
| 4 | Manual LR scheduling was explored. Sub-experiments explored different hyperparameter settings |
| 5 | Landmark features were used as input to the model |
| 6 | Further methods for data augmentation were employed to reduce overfitting |
| 7 | A Transformer architecture was trialled |
| 8 | A combination of landmark and visual features were used as input to the model |
| 9 | Letters, phonemes and visemes were assessed, respectively, to find the best classes for lip reading |

Table 4.1: A brief summary and description of the different experiments. Note that a line divides the final three experiments due to a different loss, CTC loss, being utilised.

## 4.1 Experiment 1: Base Architecture

### 4.1.1 Model Architecture

The first experiment used a basic model architecture as a basis to improve upon. The model architecture used, as shown in Figure 4.1, began with an input layer[1] to automatically configure the input shape accepted by the model. Two 1D convolutional[2] and three unidirectional LSTM layers[3] followed this. Finally a series of dense[4] and dropout layers[5] were employed before a prediction layer output the probabilities of the input sample belonging to each of the different classes within the dataset.

This architecture was inspired by Chung et al. [12] who suggested extending their architecture of CNN to additionally include LSTM units.

1D convolution was used here as the input data only consisted of image feature vectors of the size (1, 2048). ReLU activation functions were used for each layer, excluding the final prediction layer which instead employed Softmax.

Dropout layers were employed as a means of regularisation [30, 57], to avoid overfitting. A value of 0.2 was used to drop 20% of input units during training.

Categorical cross-entropy loss was used to train the model, outlined in Section 2.3.3. The classes used were the one hot encodings for the four word classes: *about*, *believe*, *chance* and *family*.

[1] https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputLayer
[2] https://keras.io/api/layers/convolution_layers/
[3] https://keras.io/api/layers/recurrent_layers/lstm/
[4] https://keras.io/api/layers/core_layers/dense/
[5] https://keras.io/api/layers/regularization_layers/dropout/

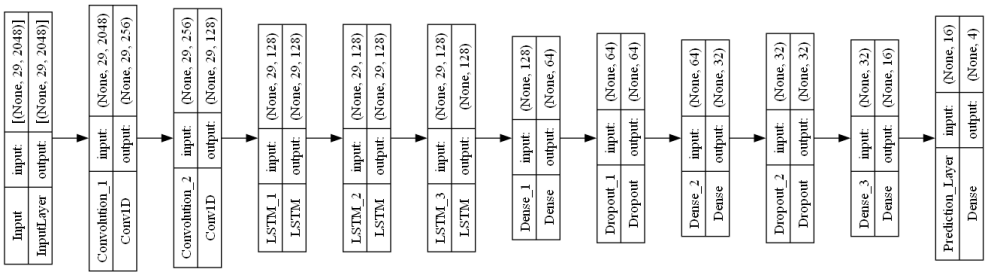



Figure 4.1: Experiment 1 architecture. This depicts two 1D convolution layers, followed by three LSTM layers and then a set of dense layers with some dropout. The "None" above refers to the batch size. Models are trained with batches of data which contain several samples of data. The 29 refers to the number of frames expected by the model. The final number for each layer is that layer's size.

The model was trained for 100 epochs to allow for sufficient time for the model to capture nuances within the training data whilst maintaining a shorter training time. A batch size of 32 was used for similar reasons.
Finally, the Adam optimiser alone was used with a LR of $1 \times 10^{-5}$. A very small LR was used because lip reading is a very nuanced and subtle process.

### 4.1.2 Results and Evaluation

The first experiment yielded promising results, as shown in Figure 4.2 and Table 4.2. Clear overfitting occurred early within training, evident by the training and testing accuracies and losses diverging early. There could be many potential causes of overfitting such as there not being enough data, data being too noisy or the model being too complex. There are many potential ways to reduce overfitting. These include data augmentation, data regularisation [30, 57] or reducing the size and complexity of the model.
Due to the results of this experiment, the second experiment instead employed some data augmentation methods, attempting to reduce noise within the data.

## 4.2 Experiment 2: Frame Pruning

### 4.2.1 Model Architecture

The same model architecture was used as in Section 4.1. However, further data preprocessing was carried out before training, aiming to reduce overfitting.
As outlined within Section 3.3.1, the structure of LRW are 29 frame videos that contain multiple word utterances. Only the primary word utterances (*about*, *believe*, *chance* or *family*) within each video are labelled. Whilst this can be beneficial, this could result in overfitting. This is because the model might fixate on commonly occurring words around the primary word, such as "I believe" rather than "believe". This could cause underfitting if the data samples are too noisy, resulting in difficulty converging to a

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 1 | 0.8394 | 1.7144 | 0.1606 |

Table 4.2: The testing accuracy, loss and WER for experiment 1. These results were captured by running the model against a testing dataset: a dataset that was not used during training of the model.

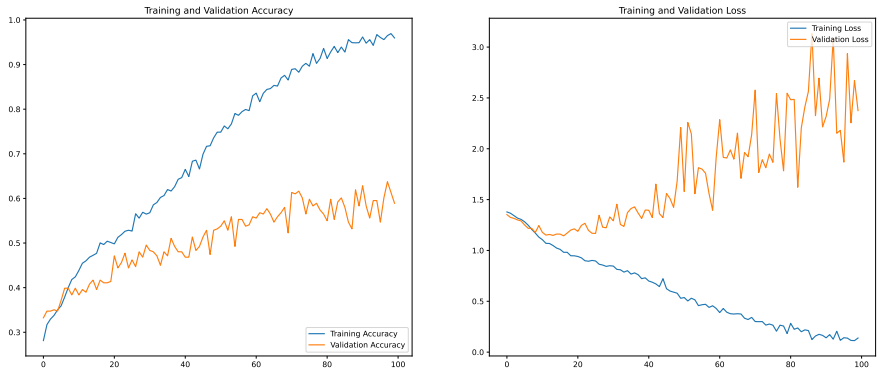



Figure 4.2: Experiment 1 results. This depicts training and testing accuracy and loss. Accuracy measures of the number of videos that were given the correct label of the word being spoken. Loss gives a measure of the categorical cross-entropy loss of the different predictions.

valid solution.

This experiment employed two methods to overcome overfitting encountered within Section 4.1: frame pruning and data augmentation.

Frame pruning was used to remove frames from the start and end of video data samples, focusing more on the central part of the video and thus the primary word utterance. This could reduce overfitting as there is less noise present in the data.

The downside of frame pruning is that it could introduce underfitting or worsen the performance of models. Indiscriminate removal of frames could remove intrinsic information from each data sample.

The method used for data augmentation was horizontal flipping. The frames for each video were horizontally flipped, and these new samples were added to the dataset. This had the effect of doubling the amount of data available for training, validation and testing. This method was used as it is one of the easiest data augmentation methods to implement [61] whilst still creating diverse enough data.

The downside of this data augmentation method is that it could lead to overfitting. After data augmentation there will be two copies of every LRW sentence, possibly worsening the issue of noisy words. If the model has even more samples of “I believe” it may become even more focused on the noisy “I”, regardless of frame flipping. This data augmentation method will preserve the structure of sentences; whilst the data might visually be different, it is not semantically diverse

Only 50 epochs were carried out for this training run, to conserve computational power.

Overfitting occurred early within the first training run and thus a high epoch number was not required to observe a similar result.

### 4.2.2 Results and Evaluation

As shown in Figure 4.3 and Table 4.3, this model performed better than the previous experiment. Early stopping helped to reduce the effect of overfitting and improve the accuracy of the model.
In conclusion, frame pruning and data augmentation did benefit the process; however, overfitting was still occurring.

## 4.3 Experiment 3: Bidirectional LSTM Architecture

### 4.3.1 Model Architecture

For this experiment, a different model architecture was investigated (see Figure 4.4). Rather than a CNN combined with an LSTM, a pure Bi-LSTM[6] architecture was studied.
Typically, convolution is utilised for image or video data (2D or 3D data) to reduce the dimensionality of the data and looking at the relationship between image regions. However, for this experiment we have already employed a visual feature extractor, effectively skipping this step.
In the previous experiment, only unidirectional LSTM layers were utilised. Instead in this experiment, we employed Bi-LSTM layers to capture both left and right context within the videos.
Training lasted for 150 epochs, allowing the model more time to train and learn the data since a simple model was being investigated. Otherwise, the details of this experiment (LR, early stopping, loss, etc) remained the same as the previous one. The same activation functions of ReLU were employed within the Bi-LSTM layers.

[6] https://keras.io/api/layers/recurrent_layers/bidirectional/

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 2 | 0.8895 | 0.4798 | 0.1105 |

Table 4.3: The testing accuracy, loss and WER for experiment 2.

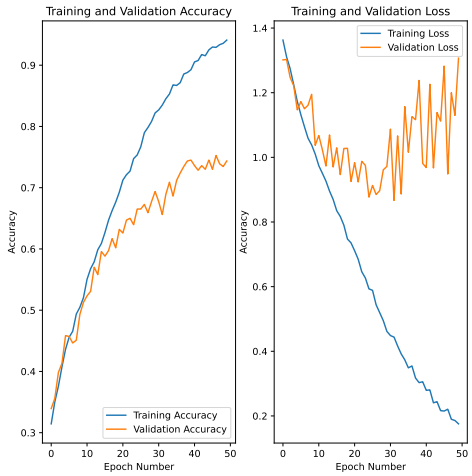



Figure 4.3: Experiment 2 results. This shows the effect of adding frame pruning and data augmentation.

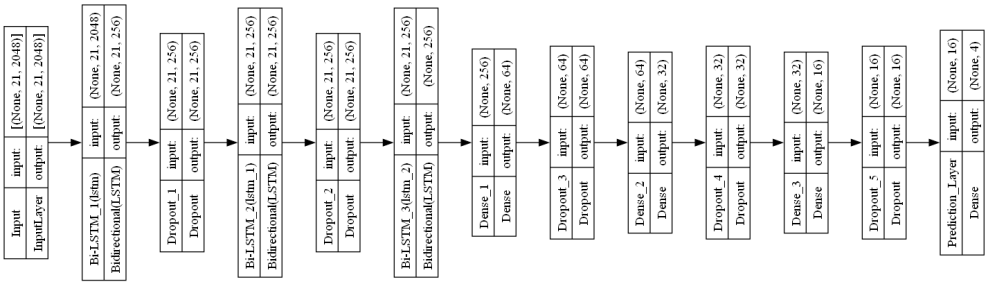



Figure 4.4: Experiment 3 architecture. This depicts the Bi-LSTM architecture employed for lip reading. The model took visual feature vectors as input to a series of Bi-LSTM layers. Dropout was employed after each layer to reduce the effect of overfitting. ReLU activation functions were utilised for each layer except the final which employed Softmax.

### 4.3.2 Results and Evaluation

As shown in Figure 4.5 and Table 4.4, this architecture actually performed worse than the previous models.
The maximum accuracy and minimum loss achieved were both lower than the models trained in the first and second experiments. However, overfitting occurred later in training, giving promising results for this architecture.
Further experimentation was conducted on this architecture aiming to produce better results.

## 4.4 Experiment 4: Manual Adaptive Learning Rate

### 4.4.1 Model Architecture

For this experiment, the same model architecture was used as in Section 4.3; however, the LR was changed.
Previously the Adam optimiser on its own had been employed to varying success. Instead, as mentioned in Section 3.4.3, a combination of both the Adam optimiser and another LR scheduling method is beneficial and thus was applied. For this experiment, Adam was used to control the LR of parameters more precisely but the LR was also altered manually.
Three primary experiments were carried out. In each, after 20 epochs the LR was multiplied by a decimal value. The values used for the experiments were 0.9, 0.5 and 0.7.
A higher initial LR of $1 \times 10^{-4}$ was used, compared with previous experiments. In most cases, a larger LR should be used initially, decreasing this over time [25]. A large initial LR allows the model to find the global minimum in the loss and then converge quickly as the LR decreases.

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 3 | 0.8998 | 1.0172 | 0.1002 |

Table 4.4: The testing accuracy, loss and WER for experiment 3.

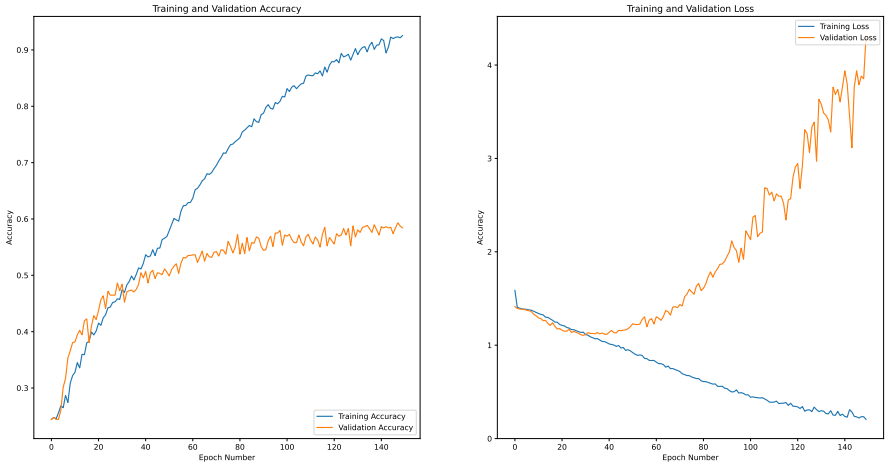
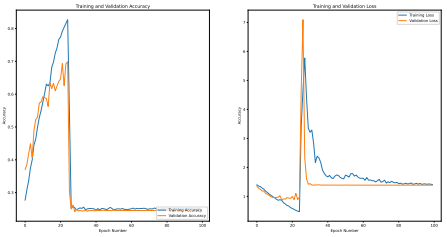
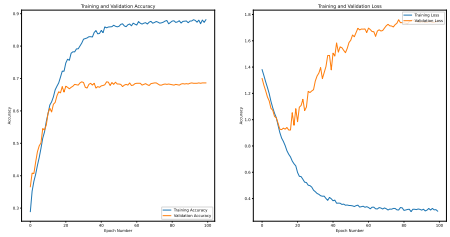


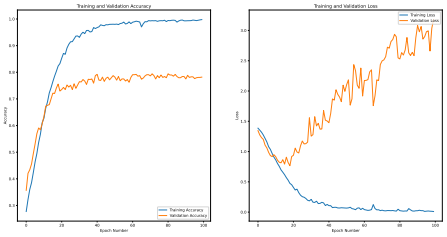

Figure 4.5: Experiment 3 results.



(a) Experiment 4.1. LR multiplier equals 0.9.



(b) Experiment 4.2. LR multiplier equals 0.5.



(c) Experiment 4.3. LR multiplier equals 0.7.

Figure 4.6: Experiment 4 results. This depicts the three experiments with adaptive LR via the Adam optimiser combined with a manual LR change via a multiplier. All three experiments used an initial LR of 0.0001 and multiplied the LR every 20 epochs. Experiment 4.1 used a decay rate of 0.9, 4.2 used 0.5 and 4.3 used 0.7.

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 4.1 | 0.8838 | 0.4490 | 0.1162 |
| 4.2 | 0.9670 | 0.3953 | 0.0330 |
| 4.3 | 0.7938 | 1.3048 | 0.2062 |

Table 4.5: The testing accuracy, loss and WER for experiment 4.

### 4.4.2 Results and Evaluation

The results of the adaptive LR experiment, represented in Figure 4.6 and Table 4.5, show that altering the LR significantly impacts the performance of models. The same model architecture but a different LR scheduling method resulted in a much-improved model, as compared with Section 4.3 which just used the Adam optimiser.
The initial experiment, which employed 0.9 for the LR multiplier, had an interesting performance. There is a large dip in performance where the model diverged and left the local minimum in the loss. The LR was potentially changed too fast in this case.
A LR multiplier of 0.5 reduced this effect, causing some overfitting but not model divergence. Moreover, the accuracy and loss with this architecture spiked far less than with previous experiments and the testing metrics gave the best results seen yet.
A LR multiplier of 0.7 strangely did not represent a middle-ground between the two experiments but instead a dip in performance. Because of this, the multiplier of 0.5 was applied to further experimentation.

## 4.5 Experiment 5: Lip Landmarks

### 4.5.1 Model Architecture

Previously, only visual features had been used for training. Alternatively, this experiment investigated the use of only landmark features as a model input.

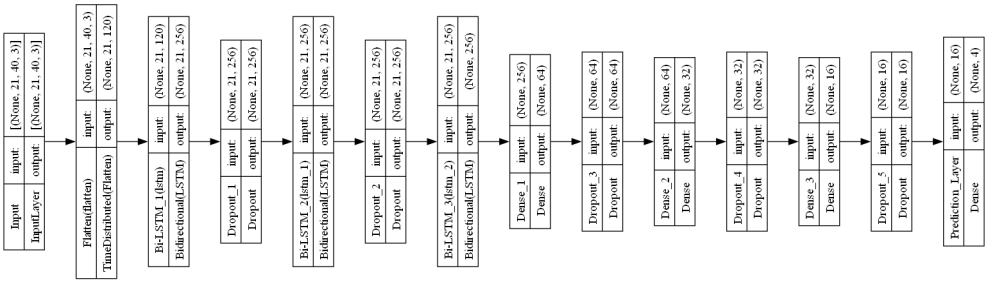


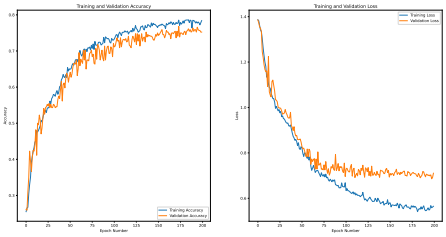

Figure 4.7: Experiment 5 architecture.

As shown in Figure 4.7, the same model architecture was used as Section 4.4 except that the input size of the model was changed. Rather than accepting 2048 long feature vectors, the model had an input size of 120. This was formed of the 40 lip landmarks, represented with $(X,Y,Z)$ coordinates, as specified in Section 3.3.2.1.
Similarly to Section 4.4, a set of sub-experiments were carried out to find the optimal performance of this architecture. The same model was used initially, to better compared with the previous experiment. The LR was then altered to find optimal performance.
Various methods were utilised to schedule and vary the LR throughout training. Whilst manual LR scheduling had already been investigated, other settings were trialled. The Adam optimiser was employed for each experiment.
Experiment 5.1 used the same LR scheduling as in experiment 4.2. Experiments 5.2 to 5.4 used different, manual LR scheduling methods and 5.5 used exponential decay, as explained in Section 3.4.3. Keras' exponential decay[7] was used to implement the decay method.

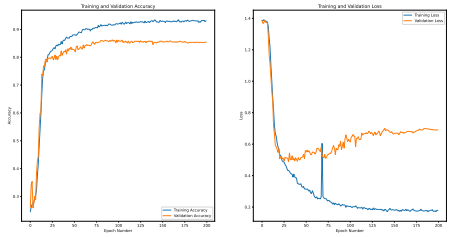
### 4.5.2 Results and Evaluation

Although experiment 5.1 did not have improved performance compared with experiment 4.2, the findings from this experiment showed that landmark features are a suitable input for training lip reading models.
The reduced performance could be related to the reduced size of the input feature vectors or due to MediaPipe's accuracy. A low probability threshold was utilised during data generation and therefore it could be that the lip landmarks were less accurate than desired. This problem is not present when using visual features, which are just crops of the mouth region.
After this finding, different values for the LR scheduling were assessed to optimise this model further, the results for which are represented within Figure 4.8 and Table 4.6.
The next four experiments showed the impact of tuning the scheduler's parameters, significantly altering the performance of the model. When manually altering the LR, starting with a higher initial LR and a larger decay rate (leading to smaller changes in the LR) performed better. Furthermore, Exponential decay performed very well, producing one of the best models in this research yet.
The LR setting presented in experiment 5.4 performed the best. However, the LR

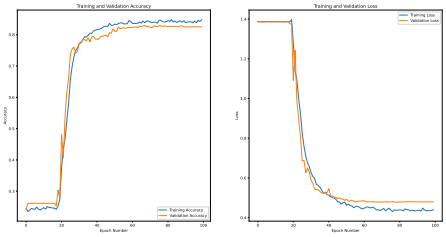
[7] https://keras.io/api/optimizers/learning_rate_schedules/exponential_decay/



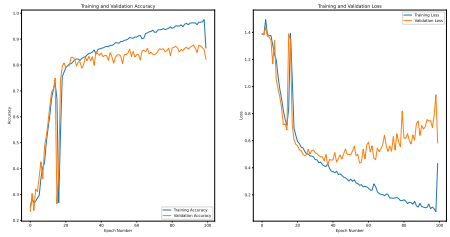
(a) Experiment 5.1.



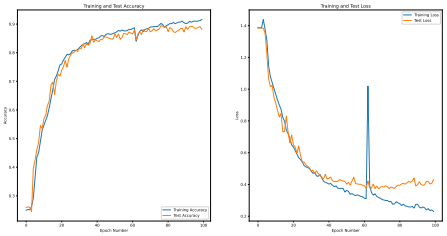
(b) Experiment 5.2.



(c) Experiment 5.3.



(d) Experiment 5.4.



(e) Experiment 5.5.

Figure 4.8: Experiment 5 results. This depicts experiments 5.1, 5.2, 5.3, 5.4 and 5.5. Each of these experiments utilised the same model architecture but varying LR scheduling methods. 5.1–5.4 utilised manual methods of changing the LR whilst 5.5 showcased exponential decay.

| Experiment | LR Configuration | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|---|
| 5.1 | **Scheduler**: Manual<br>**Initial LR**: 0.0001<br>**Change steps**: 20<br>**LR change**: 0.5 | 0.8189 | 0.5370 | 0.1811 |
| 5.2 | **Scheduler**: Manual<br>**Initial LR**: 0.001<br>**Change steps**: 20<br>**LR change**: 0.5 | 0.9567 | 0.1374 | 0.0433 |
| 5.3 | **Scheduler**: Manual<br>**Initial LR**: 0.001<br>**Change steps**: 30<br>**LR change**: $e^{-0.1}$ | 0.8656 | 0.3540 | 0.1344 |
| 5.4 | **Scheduler**: Manual<br>**Initial LR**: 0.001<br>**Change steps**: 30<br>**LR change**: $e^{-0.01}$ | 0.9704 | 0.0949 | 0.0296 |
| 5.5 | **Scheduler**: Exponential decay<br>**Initial LR**: 0.001<br>**Decay steps**: 275<br>**Decay rate**: 0.9 | 0.9305 | 0.2111 | 0.0695 |

Table 4.6: The testing accuracy, loss and WER for experiment 5. This shows the settings for each of the different sub-experiments and the resulting testing accuracy and loss.

scheduler from experiment 5.5 was taken forwards. Although this model had worse testing metrics, the spiking within the accuracy and loss was far decreased compared with the other methods. Additionally, overfitting appeared to have been diminished. This reliability is more useful than hyperparameters configured for a specific experiment. Experiment 5.5's LR scheduling is more general and thus will be carried forward, rather than the settings for experiment 5.5.

## 4.6 Experiment 6: Data Augmentation

### 4.6.1 Model Architecture

In Section 4.5, although overfitting was reduced, there was still divergence among the loss and accuracy in the later epochs. Thus, further methods of data augmentation were assessed to reduce this effect.

The method of data augmentation that was employed was a set of random rotations. Each of the data samples were randomly rotated and the new samples were added again to the dataset, to bolster the amount of training data. The same rotation was applied to each lip landmark within a frame, and each frame of a given video. The rotations were in the range $[-\frac{\pi}{8}, -0.1]$ and $[0.1, \frac{\pi}{8}]$. This was to ensure that the generated samples were not too similar to the originals. Normally, viewers will be upright and not rotated, so rotations were kept minimal.

Otherwise, the architecture for this experiment was identical to experiment 5.5.

### 4.6.2 Results and Evaluation

Shown in Figure 4.9 and Table 4.7, the results of this experiment reflected reduced performance, showing that the data augmentation was not beneficial. This may be due to the random rotations not being drastic enough or the amount of data augmentation being too much.

Previous to this experiment, each data sample was technically in the dataset twice

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 6 | 0.8807 | 0.4006 | 0.0490 |

Table 4.7: The testing accuracy, loss and WER for experiment 6.

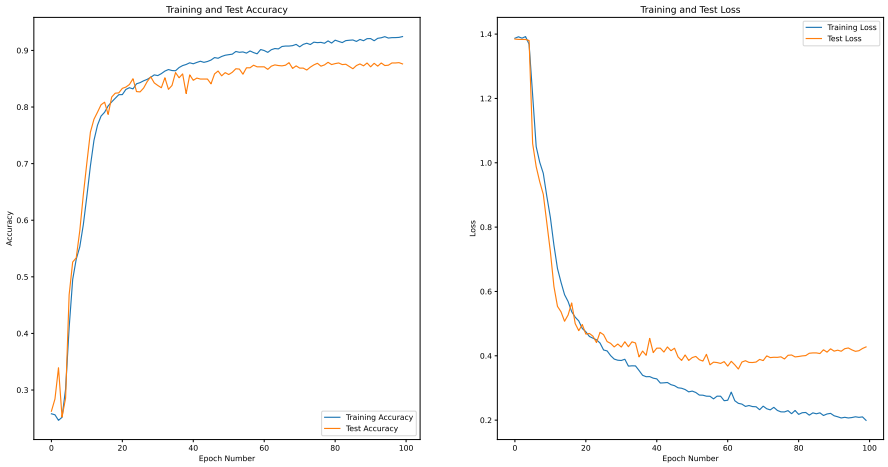



Figure 4.9: Experiment 6 results. This shows the impact of a further set of data augmentations: landmark rotations. The result of this shows decreased performance and a more overfit model.

already: having been augmented once. Random rotations meant that single data samples, although augmented, were presented to the model up to four times. Consequently, overfitting could have been due to the structure of the sentences within the data samples, and repeated samples with these structures. Repeated noise might have been causing overfitting. Consequently, this extra data augmentation method was not applied to further experiments.

## 4.7 Experiment 7: Transformer Architecture

### 4.7.1 Model Architecture

For this experiment a different model architecture was used, shown in Figure 4.10. The mechanism of attention, explained in Section 2.3.6, was employed to construct an encoder Transformer architecture.

Landmark features were input and combined with a positional encoding layer[8]. The output of this was fed into a series of encoder blocks, made up of multi-head attention[9] and *Add&Norm* layers[10]. The output from this was then fed into a simple feed-forward

[8] https://keras.io/api/layers/core_layers/embedding/

[9] https://keras.io/api/layers/attention_layers/multi_head_attention/

[10] https://keras.io/api/layers/normalization_layers/layer_normalization/

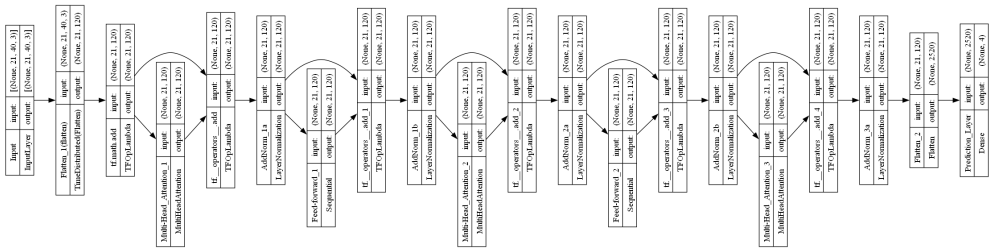


Figure 4.10: Experiment 7 architecture. This comprised of an input layer accepting landmark features followed by a series of three encoder blocks. Encoder blocks were made up of multi-head attention blocks and *Add&Norm* layers. Finally, a feed-forward network was used at the end, before the final hidden representation was fed into a prediction layer.

network and then a dense layer for classification. The purpose of this experiment was to investigate the potential benefits of attention and a Transformer architecture for lip reading.

The same LR scheduling and batch size was used as experiment 5.5.

### 4.7.2 Results and Evaluation

Shown in Figure 4.11 and Table 4.8, the resulting model performed worse than previous models. Overfitting occurred but, as a proof of concept, the Transformer architecture for lip reading was highly successful. Therefore, further experimentation was carried out to assess this architecture further.

## 4.8 Experiment 8: Image and Landmarks

### 4.8.1 Model Architecture

For this experiment, the Transformer architecture was adapted to improve performance. Inspired by Xue et al. [76], both visual and landmark features were utilised as inputs to a model, to investigate their potential combined benefit.

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 7 | 0.8724 | 0.3437 | 0.1276 |

Table 4.8: The testing accuracy, loss and WER for experiment 7.

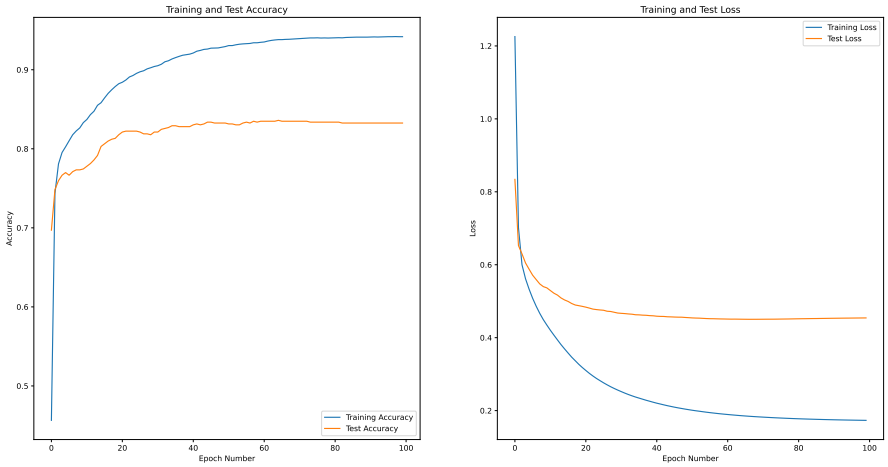



Figure 4.11: Experiment 7 results

As depicted within Figure 4.12, visual and landmark features were fed into independent self-attention blocks. These blocks consisted of dot-product self-attention[11] and *Add&Norm* layers[12]. The two streams were then fed into similar cross-attention blocks. These blocks instead used the query from one input stream and the key from the alternate stream, hence comparing the two streams and, therefore, formulating cross-attention. The output from this was passed through a further *Add&Norm* layer before being input into a CNN. Xue et al. utilised a series of GRU units but our research found that convolutional layers performed better.
The same LR scheduler was used as in experiment 5.5 for better comparison of the new model architecture.

### 4.8.2 Results and Evaluation

As depicted in Figure 4.13 and Table 4.9, this model architecture produced one of the best performances for lip reading yet. Utilising both landmark and visual features with

[11] https://keras.io/api/layers/attention_layers/attention/
[12] https://keras.io/api/layers/normalization_layers/layer_normalization/

| Experiment | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|
| 8 | 0.9203 | 0.2730 | 0.0797 |

Table 4.9: The testing accuracy, loss and WER for experiment 8.

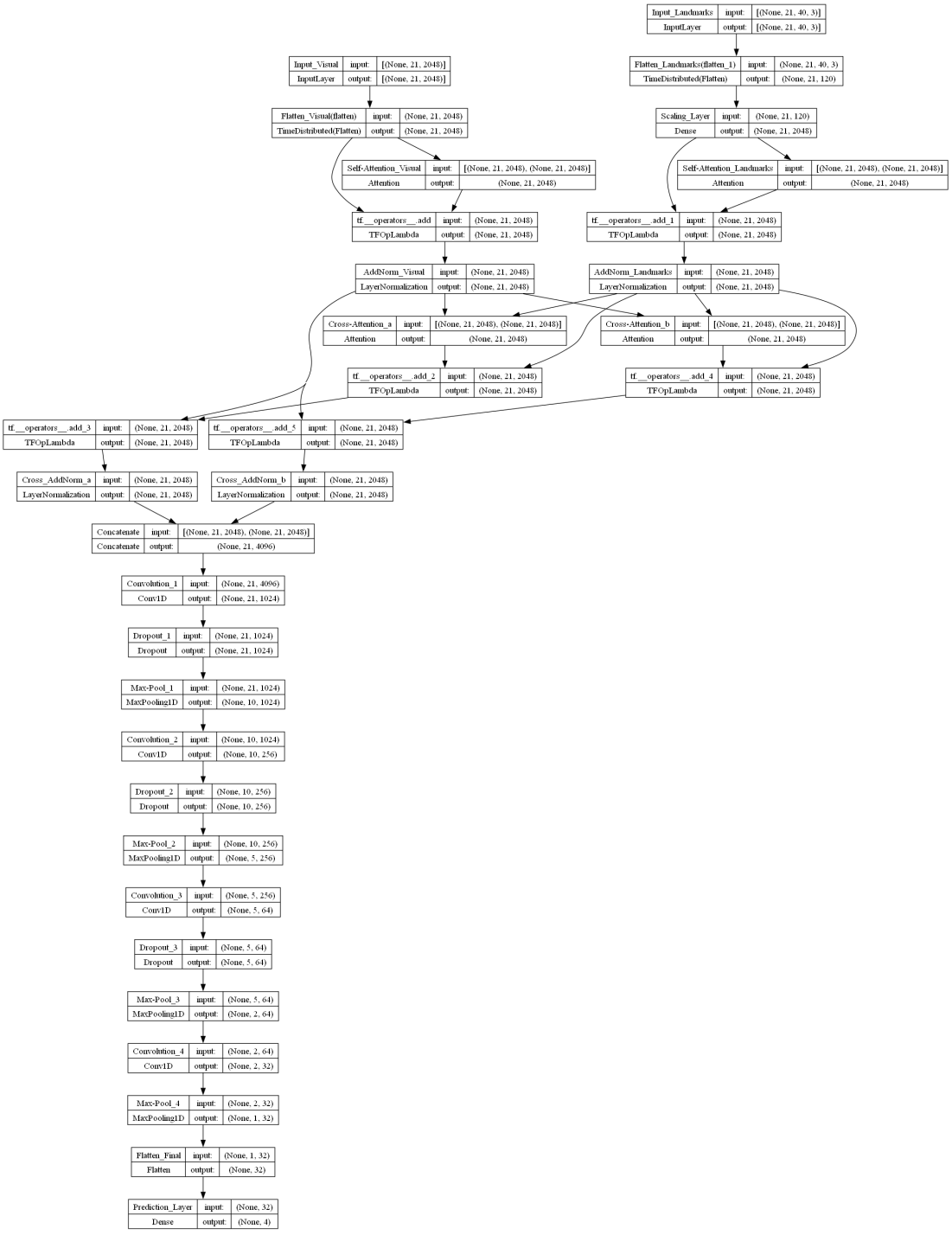



Figure 4.12: Experiment 8 architecture. This architecture was based on the work of Xue et al. [76]. The model took both visual and landmark features as input to an encoder. The encoder employed self-attention and *Add&Norm* on either input before using cross-attention between them. The output was then fed into a series of convolutional layers and the prediction layer.

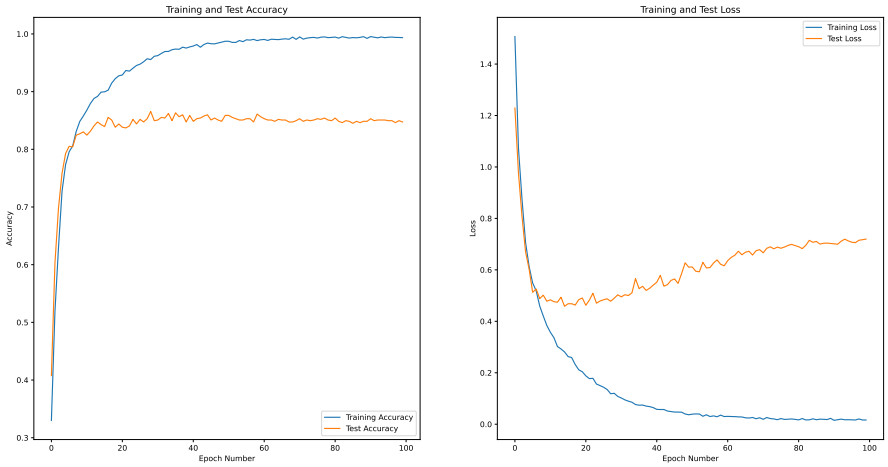



Figure 4.13: Experiment 8 results

this Transformer architecture provides another, different solution for lip reading.
The performance of this model was not as good as experiment 5.4 and so further experimentation was carried out.

## 4.9 Experiment 9: CTC Loss

### 4.9.1 Model Architecture

For this experiment, the same model architecture was used as experiment 8, however, the classes and loss metric were changed.
Outlined in Section 2.3.8, CTC loss is a useful loss metric capable of drawing connections between two sequences when this alignment is unknown [45]. This fits the task of aligning a set of letters, phonemes or visemes to a sequence of video frames.
This experiment focused on experimenting with CTC loss to make three different models capable of predicting the letter, phoneme and viseme uttered in each frame.
Because a different loss metric has been used to train these models, they cannot be directly compared to the previous experiments. Different loss metrics mean that we must independently inspect these models and their performance with lip reading.
Different class labels were used for these experiments, changing from single-label classification to multi-label classification. For experiment 9.1 these were the letters for the word uttered, experiment 9.2 used the phonemes and experiment 9.3 used the visemes.

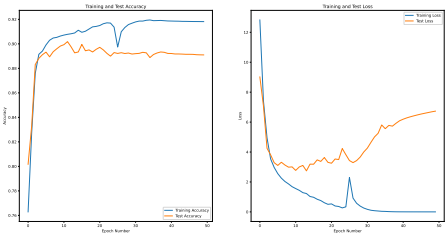
This transformed the classes from 1D one hot encodings to instead 1D vectors of class numbers.
The same LR scheduling method was used as in experiment 5.5.

### 4.9.2 Results and Evaluation

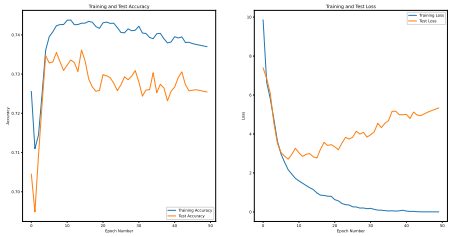
The findings of this experiment, presented within Table 4.10 and Figure 4.14, are incredibly interesting. As suspected, the best class for lip reading was indeed viseme-based, followed by phoneme-based and finally letter-based. This is intuitive as many letters in the English language are not pronounced or are pronounced differently depending on the context. For example, the “h” in “her” and “there” are very different.
Phoneme-based lip reading presented the worst accuracy but better loss and WER compared with the other methods. A potential reason for this may be some phonemes not being visually represented. Especially in the English lexicon, many sounds are spoken quickly, skipped or pronounced without changes to the facial position. Consequently, the model may struggle to distinguish some words based on their phonemes.
The finding that viseme-based learning is useful for lip reading is incredibly important and shows the impact of focusing on visual aspects of speech. Furthermore, this proof of concept shows that many visual-only lip reading systems could be improved and made more generic and able to process far more words by looking at their sub-words, particularly their visemes.

| Experiment | Class Types | Testing Accuracy | Testing Loss | WER |
|---|---|---|---|---|
| 9.1 | Letter | 0.9076 | 2.6221 | 0.2187 |
| 9.2 | Phoneme | 0.7412 | 1.9542 | 0.2107 |
| 9.3 | Viseme | 0.8068 | 1.7748 | 0.1765 |

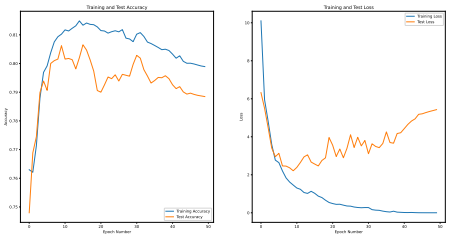
Table 4.10: The testing accuracy, loss and WER for experiment 9. Three different sub-experiments were conducted, using CTC loss, to compare using the letters, phonemes and visemes as classes for lip reading.



(a) Letter based model



(b) Phoneme based model



(c) Viseme based model

Figure 4.14: Experiment 9 results. Three different sub-experiments were conducted, using CTC loss, to compare using the letters, phonemes and visemes as classes for lip reading.

# Chapter 5

# Conclusion

## 5.1 Summary of Achievements

| Experiment | Summary | Val Accuracy | Val Loss | WER |
|---|---|---|---|---|
| 4.2 | Visual-based | 0.9670 | 0.3953 | 0.0330 |
| 5.4 | Landmark-based Bi-LSTM | 0.9704 | 0.0949 | 0.0296 |
| 8 | Landmark, Visual Transformer | 0.9203 | 0.2730 | 0.0797 |
| 9.1 | Letter-based with CTC loss | 0.9076 | 2.6221 | 0.2187 |
| 9.2 | Phoneme-based with CTC loss | 0.7412 | 1.9542 | 0.2107 |
| 9.3 | Viseme-based with CTC loss | 0.8068 | 1.7748 | 0.1765 |

Table 5.1: The final models presented for lip reading. This shows the testing accuracy, loss and WER achieved for the best lip reading models trained within this research. There is slight variation within the design, architecture and training of these models, outlined within the relevant sections above.

In this research, we aimed to compare different methods of using Machine Learning and Computer Vision for visual-only lip reading.
We started by developing a data generation pipeline to preprocess data from the BBC into a format that could be trained upon. The LRW dataset was selected due to its reliability and because it suited the independent word recognition task studied here.
Various architectures were trained, compared, and modified to create the best lip reading solution. First, simple architectures such as Convolutional Neural Network (CNN) and Bidirectional Long Short Term Memory (Bi-LSTM) networks were investigated, but later more complex Transformer architectures were used to great success. The Learning Rate (LR) was significantly varied and exploited to create the best possible models. From this research, it was discovered that Transformer architectures and

Bi-LSTM networks performed the best for lip reading. It was also found that a combination of landmark and visual inputs can improve lip reading performance.
Finally, Connectionist Temporal Classification (CTC) loss was used to train lip reading ANNs capable of processing each independent frame, distinguishing each sub-word and predicting the letter, phoneme or viseme uttered. These models were compared to find the best solution for frame-to-frame lip reading which was, as hypothesised, viseme-based prediction.
Overall, seventeen experiments were conducted. A summary of the best models developed for lip reading is displayed in Table 5.1, displaying up to 97% accuracy, less than 10% loss and a Word Error Rate (WER) of just 13.7%. A summary of all of the results of these experiments is shown in Table A.1.
Overall, all of the objectives established in Section 1.4 were achieved. An efficient, configurable and automatic data generation pipeline was developed, making it easier to extend this work in the future. Various ANN architectures were designed, trained and objectively compared to find the best solution for lip reading. A Graphical User Interface (GUI) was also developed to showcase the different models and alter the processing of their outputs. Additionally, the GUI's extensional aim was achieved by incorporating model fine-tuning, which improved lip reading performance for specific users.

## 5.2 Critical Reflection

As with any project, it is crucial to reflect upon its inherent strengths and weaknesses. This reflection aims to highlight potential areas of improvement whilst implementing lip reading using Machine Learning (ML).
One major limitation of this work was the size of the data involved. Much of the training and experimentation was limited to a small subset of words. This decision was taken to shorten the potential long training times, reduce the required data storage and avoid some difficult issues such as with high-dimensional classification. The small word set allowed further focus on the task at hand. Whilst the findings from this experiment were conducted with rigorous adherence to the scientific method, and with a large amount of data, further words could have been incorporated to offer better reliability, additional comparison of model performance and create a more general model suitable for wider application. Future improvements could extend the vocabulary, trying a larger set of words and potentially comparing sub-word and word-level

classification. This work could take the achievements found with the Transformer architecture and CTC loss, extending them to a wider dataset.
Another limitation of this work was the intrinsic nature of model evaluation. Testing and evaluation were primarily completed using data from the LRW dataset, with some testing via the GUI. Extrinsic evaluation, on completely unseen and different data, would give a better understanding of the generalisability, application and true performance of the lip reading models. All data within the LRW dataset is similar, making the analysis less reliable. Future work could validate these models against external data from a setting that is completely unseen. This would give additional information as to the usefulness of models and provide further insight as to methods for further improving model performance.
Finally, due to timing constraints, and it not being the true focus of the project, the GUI was not as usable or aesthetic as it could have been. The aims of this project were primarily around lip reading research and model comparison, rather than developing an efficient application. Further time could extend this GUI into a more usable application, with user evaluation even being carried out to maximise the user experience of a formal lip reading application.

## 5.3 Future Work

The research conducted in this study offers numerous avenues for potential applications. This section will explain some potential extensions or uses of the work.
Firstly, and the most obvious, would be to extend the experiments conducted within this research. Models could be trained to recognise wider word sets, different model architectures could be trialled or further parameter settings could be assessed. Continually, new architectures are found that perform better in specific situations. Creating different model architectures, training them for lip reading and comparing their performance would help to extend the research further. The experiments above have various parameters that could be investigated in more depth.
Some suggestions for variations on experiments include:

- During data preprocessing, utilise a larger set of MediaPipe landmarks than the forty selected for this research. An entirely different facial landmark system could be investigated, such as the solution provided by DLib
- During data preprocessing, crop a larger region of the mouth or face

- Compare different visual feature extractors and their performance during training
- Expand the word set for training
- Expand testing to an unseen task or different dataset
- Train a specific visual feature extractor for human faces, rather than employing a pretrained extractor

Next, whilst this approach was research-centered, further work could use the findings here to develop an application built around lip reading. Hardware could specifically be made to help people with hearing loss on the go. For example, a pair of glasses could be made to employ lip reading in real-time. Uni-directional microphones could be used to focus on a single person and collect audio to be paired with visual features. This technology could improve closed caption generation, could generate audio in a visual-only medium or improve security. Many security cameras are outdated, giving no audio, so such a model could help law enforcement to listen to criminal activity from such cameras.

In summary, there are various ways this research could be extended. Lip reading is crucial to everyday communication and Machine Learning presents a new way forwards that could greatly improve audio processing, give insight into teaching lip reading and help people with Disabling Hearing Loss (DHL).

# Bibliography


[1] M. Abdullah, M. Ahmad, and D. Han. Facial expression recognition in videos: An cnn-lstm based model for video classification. In *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–3, 2020.

[2] D. Adams. *The Hitchhiker's Guide to the Galaxy*. New York: Harmony Books, 1980.

[3] T. Afouras, J. S. Chung, and A. Zisserman. Lrs3-ted: a large-scale dataset for visual speech recognition. *arXiv preprint arXiv:1809.00496*, 2018.

[4] S. Agrawal, V. R. Omprakash, et al. Lip reading techniques: A survey. In *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pages 753–757. IEEE, 2016.

[5] K. Banerjee, R. R. Gupta, K. Vyas, B. Mishra, et al. Exploring alternatives to softmax function. *arXiv preprint arXiv:2011.11538*, 2020.

[6] H. L. Bear, R. W. Harvey, B.-J. Theobald, and Y. Lan. Which phoneme-to-viseme maps best improve visual-only computer lip-reading? In *Advances in Visual Computing: 10th International Symposium, ISVC 2014, Las Vegas, NV, USA, December 8-10, 2014, Proceedings, Part II 10*, pages 230–239. Springer, 2014.

[7] E. Bisong and E. Bisong. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64, 2019.

[8] L. Cahill and C. Tiberius. Cross linguistic phoneme correspondences. In *COLING 2002: The 17th International Conference on Computational Linguistics: Project Notes*, 2002.

[9] R. Campbell. The processing of audio-visual speech: empirical and neural bases. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 363(1493):1001–1010, 2008.

[10] R. Chauhan, K. K. Ghanshala, and R. Joshi. Convolutional neural network (CNN) for image detection and recognition. In *2018 first international conference on secure cyber computing and communication (ICSCCC)*, pages 278–282. IEEE, 2018.

[11] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[12] J. S. Chung and A. Zisserman. Lip reading in the wild. In *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II 13*, pages 87–103. Springer, 2017.

[13] L. L. Cunningham and D. L. Tucci. Hearing loss in adults. *New England Journal of Medicine*, 377(25):2465–2473, 2017. PMID: 29262274.

[14] A. Devarakonda, M. Naumov, and M. Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *CoRR*, abs/1712.02029, 2017.

[15] A. Dhillon and G. K. Verma. Convolutional neural network: a review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, 9(2):85–112, 2020.

[16] R. DiPietro and G. D. Hager. Deep learning: RNNs and LSTM. In *Handbook of medical image computing and computer assisted intervention*, pages 503–519. Elsevier, 2020.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[18] N. P. Erber. Interaction of audition and vision in the recognition of oral speech stimuli. *Journal of speech and hearing research*, 12(2):423–425, 1969.

[19] S. Geylanioglu. Developing English language learners' pronunciation through conceptualization, 06 2017.

[20] H. Gholamalinezhad and H. Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

[21] B. Ghosh, I. K. Dutta, A. Carlson, M. Totaro, and M. Bayoumi. An empirical analysis of generative adversarial network training times with varying batch sizes. In *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0643–0648. IEEE, 2020.

[22] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[23] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[24] M. A. Islam. Reduced dataset neural network model for manuscript character recognition. 2020.

[25] I. Kandel and M. Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020.

[26] N. Ketkar and N. Ketkar. Stochastic gradient descent. *Deep learning with Python: A hands-on introduction*, pages 113–132, 2017.

[27] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[28] D. E. King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009.

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

[30] J. Kukacka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. *CoRR*, abs/1710.10686, 2017.

[31] I. Kulikovskikh, S. Prokhorov, T. Legović, and T. Šmuc. An sgd-based meta-learner with "growing" descent. In *Journal of Physics: Conference Series*, volume 1368, page 052008. IOP Publishing, 2019.

[32] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[33] J. Lee and S. Watanabe. Intermediate loss regularization for ctc-based speech recognition. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6224–6228, 2021.

[34] S. Lee and D. Yook. Audio-to-visual conversion using hidden markov models. In *Pacific Rim International Conference on Artificial Intelligence*, pages 563–570. Springer, 2002.

[35] S. Leixian, Q. Zhang, G. Cao, and H. Xu. Fall detection system based on deep learning and image processing in cloud environment. pages 590–598, 01 2019.

[36] H. Li, J. Li, X. Guan, B. Liang, Y. Lai, and X. Luo. Research on overfitting of deep learning. In *2019 15th International Conference on Computational Intelligence and Security (CIS)*, pages 78–81, 2019.

[37] R. J. Lickley. Fluency and disfluency. *The handbook of speech production*, pages 445–474, 2015.

[38] Y. Liu, Y. Zhang, Y. Wang, F. Hou, J. Yuan, J. Tian, Y. Zhang, Z. Shi, J. Fan, and Z. He. A survey of visual transformers. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[39] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. Yong, J. Lee, et al. Mediapipe: A framework for perceiving and processing reality. In *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)*, volume 2019, 2019.

[40] F. Lundh. An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[41] M. Luo, S. Yang, X. Chen, Z. Liu, and S. Shan. Synchronous bidirectional learning for multilingual lip reading. *CoRR*, abs/2005.03846, 2020.

[42] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[43] D. K. Margam, R. Aralikatti, T. Sharma, A. Thanda, P. A. K, S. Roy, and S. M. Venkatesan. Lipreading with 3d-2d-cnn BLSTM-HMM and word-ctc models. *CoRR*, abs/1906.12170, 2019.

[44] L. S.-T. Memory. Long short-term memory. *Neural computation*, 9(8):1735–1780, 2010.

[45] Y. Miao, M. Gowayyed, X. Na, T. Ko, F. Metze, and A. Waibel. An empirical exploration of CTC acoustic models. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2623–2627, 2016.

[46] W. K. Mutlag, S. K. Ali, Z. M. Aydam, and B. H. Taher. Feature extraction methods: a review. In *Journal of Physics: Conference Series*, volume 1591, page 012028. IOP Publishing, 2020.

[47] Z. Niu, G. Zhong, and H. Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

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

[49] P. Parnes, D. Cameron, N. Christie, L. Cockburn, G. Hashemi, and K. Yoshida. Disability in low-income countries: Issues and implications. *Disability and rehabilitation*, 31(14):1170–1180, 2009.

[50] J. Perkins. *Python 3 text processing with NLTK 3 cookbook*. Packt Publishing Ltd, 2014.

[51] R. R. Picard and K. N. Berk. Data splitting. *The American Statistician*, 44(2):140–147, 1990.

[52] P. Podrzaj. A brief demonstration of some python gui libraries. In *Proceedings of the 8th International Conference on Informatics and Applications ICIA2019*, pages 1–6, 2019.

[53] P. M. Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. 2017.

[54] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[55] A. Rao. Transcribing educational videos using whisper: A preliminary study on using ai for transcribing educational videos. *arXiv preprint arXiv:2307.03200*, 2023.

[56] Z. Reitermanova et al. Data splitting. In *WDS*, volume 10, pages 31–36. Matfyzpress Prague, 2010.

[57] C. F. G. D. Santos and J. P. Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Computing Surveys (CSUR)*, 54(10s):1–25, 2022.

[58] J. E. Saunders, D. M. Barrs, W. Gong, B. S. Wilson, K. Mojica, and D. L. Tucci. Cost effectiveness of childhood cochlear implantation and deaf education in nicaragua: a disability adjusted life year model. *Otology & Neurotology*, 36(8):1349–1356, 2015.

[59] G. Schwiebert, C. Weber, L. Qu, H. Siqueira, and S. Wermter. A multimodal german dataset for automatic lip reading systems and transfer learning. *arXiv preprint arXiv:2202.13403*, 2022.

[60] D. Shah. Cross entropy loss: Intro, applications, code, 2023.

[61] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

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

[63] J. Son Chung, A. Senior, O. Vinyals, and A. Zisserman. Lip reading sentences in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6447–6456, 2017.

[64] B. Spehar, S. Goebel, and N. Tye-Murray. Effects of context type on lipreading and listening performance and implications for sentence processing. *Journal of speech, language, and hearing research*, 58(3):1093–1102, 2015.

[65] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[66] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng. Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning*, pages 10183–10192. PMLR, 2021.

[67] J. Terra. Keras vs tensorflow vs pytorch: Key differences among deep learning, 2023.

[68] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[69] G. Vyas. The processing of audio-visual speech: empirical and neural bases, 2023.

[70] L. Woodhouse, L. Hickson, and B. Dodd. Review of visual speech perception by hearing and hearing-impaired people: clinical implications. *International Journal of Language & Communication Disorders*, 44(3):253–270, 2009.

[71] World Health Organization. Addressing the rising prevalence of hearing losss, 2018.

[72] World Health Organization. Deafness and hearing loss, 2023.

[73] H. Wu and X. Gu. Max-pooling dropout for regularization of convolutional neural networks. In *Neural Information Processing: 22nd International Conference, ICONIP 2015, Istanbul, Turkey, November 9-12, 2015, Proceedings, Part I 22*, pages 46–54. Springer, 2015.

[74] K. Xu, D. Li, N. Cassimatis, and X. Wang. Lcanet: End-to-end lipreading with cascaded Attention-CTC. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 548–555, 2018.

[75] Z. Xu, A. M. Dai, J. Kemp, and L. Metz. Learning an adaptive learning rate schedule. *arXiv preprint arXiv:1909.09712*, 2019.

[76] F. Xue, Y. Li, D. Liu, Y. Xie, L. Wu, and R. Hong. Lipformer: Learning to lipread unseen speakers based on visual-landmark transformers, 2023.

[77] S. Yang, Y. Zhang, D. Feng, M. Yang, C. Wang, J. Xiao, K. Long, S. Shan, and X. Chen. LRW-1000: A naturally-distributed large-scale benchmark for lip reading in the wild. In *2019 14th IEEE international conference on automatic face & gesture recognition (FG 2019)*, pages 1–8. IEEE, 2019.

[78] X. Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[79] K. You, M. Long, M. I. Jordan, and J. Wang. Learning stages: Phenomenon, root cause, mechanism hypothesis, and implications. *CoRR*, abs/1908.01878, 2019.

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

[81] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[82] J. Zhang, L. Zi, Y. Hou, D. Deng, W. Jiang, and M. Wang. A c-BiLSTM approach to classify construction accident reports. *Applied Sciences*, 10(17), 2020.

# Appendix A

# Model Metric Summary

| Experiment | Validation Accuracy | Validation Loss | WER |
|---|---|---|---|
| 1 | 0.8394 | 1.7144 | 0.1606 |
| 2 | 0.8895 | 0.4798 | 0.1105 |
| 3 | 0.8998 | 1.0172 | 0.1002 |
| 4.1 | 0.8838 | 0.4490 | 0.1162 |
| 4.2 | 0.9670 | 0.3953 | 0.0330 |
| 4.3 | 0.7938 | 1.3048 | 0.2062 |
| 5.1 | 0.8189 | 0.5370 | 0.1811 |
| 5.2 | 0.9567 | 0.1374 | 0.0433 |
| 5.3 | 0.8656 | 0.3540 | 0.1344 |
| 5.4 | 0.9704 | 0.0949 | 0.0296 |
| 5.5 | 0.9305 | 0.2111 | 0.0695 |
| 6 | 0.8807 | 0.4006 | 0.0490 |
| 7 | 0.8724 | 0.3437 | 0.1276 |
| 8 | 0.9203 | 0.2730 | 0.0797 |
| 9.1 | 0.9076 | 2.6221 | 0.2187 |
| 9.2 | 0.7412 | 1.9542 | 0.2107 |
| 9.3 | 0.8068 | 1.7748 | 0.1765 |

Table A.1: Comparative metrics for all models trained. This shows the validation accuracy, loss and WER (where applicable) for all different trained models. Note that lines are used between models that cannot be compared either due to a different training data, loss metrics or otherwise.